Univerzita Komenského v Bratislave Fakulta matematiky,
fyziky a informatiky

Predpovedanie času riešenia pre ťažké logické úlohy

Diplomová práca

2019
Bc. Lukáš Ivan

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

Predpovedanie času riešenia pre ťažké logické úlohy

Diplomová práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra informatiky
Školiteľ:	RNDr. Michal Forišek, PhD.

Bratislava, 2019
Bc. Lukáš Ivan

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Lukáš Ivan
Študijný program: informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský
Sekundárny jazyk: anglický

Názov: Predpovedanie času riešenia pre ťažké logické úlohy
Predicting the solving time for hard logic puzzles

Anotácia: Práca sa zaoberá skúmaním, od akých faktorov závisí priemerný čas, ktorý potrebuje ľudský riešiteľ na vyriešenie konkrétnej inštancie algoritmicky ťažkej logickej úlohy.

Cieľ: Hlavným cieľom práce je skúmanie otázky, ako ľudia riešia konkrétne inštancie ťažkých (zväčša NP-ťažkých alebo PSPACE-ťažkých) logických úloh. Zaujímavá je otázka, ktoré faktory vplývajú na priemerný čas potrebný na vyriešenie konkrétnej inštancie a špeciálne to, či a ako dobre vieme predpovedať čas potrebný na riešenie inštancie zo syntaktických vlastností danej inštancie (ako napríklad veľkosť či "tvar" priestoru stavov). Súčasťou práce by mala byť analýza dostatočne veľkej vzorky reálnych dát o ľudských riešiteľoch.

Vedúci: RNDr. Michal Forišek, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: prof. RNDr. Martin Škoviera, PhD.

Dátum zadania: 12.12.2017

Dátum schválenia: 13.12.2017 prof. RNDr. Rastislav Kráľovič, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Poďakovanie: Týmto by som chcel poďakovať školiteľovi mojej diplomovej práce RNDr. Michalovi Forišekovi, PhD. za jeho odborné vedenie, metodickú pomoc a cenné rady, ktoré mi poskytol pri jej vypracovaní.

Abstrakt

Cieľom práce je skúmanie spôsobu, akým ľudia riešia konkrétne výpočtovo ťažké logické úlohy a analýza ako syntaktické vlastnosti inštancií problémov ovplyvňujú obtiažnosť danej inštancie. Riešenie problematiky má základ v analýze stavového grafu vytvoreného z konkrétnych inštancií problému. V práci sa zaoberáme logickou úlohou Rush Hour patriacou do množiny PSPACE-úplných problémov, pre ktorú sme vytvorili webový portál na zber reálnych dát z riešenia úloh ľudským riešiteľom. Na účely analýzy stavových grafov sme vytvorili program na vykresľovanie a manipuláciu 3D grafov stavového priestoru. Zahŕňa aj grafy stavového priestoru vážené dátami z portálu, návštevami a časom pobytu riešiteľov v grafových prvkoch. Na predpovedanie času riešenia sme dáta z portálu spolu s externými dátami ďalej analyzovali pomocou modelov strojového učenia. Pre zjednodušenie predikcie obtiažnosti sme vybrali len niektoré vlastnosti stavového priestoru. Metódou prehľadávania mriežky sme získali najlepšie modely a následne sme urobili extrakciu najvýznamnejších vlastností pomocou metódy rekurzívneho eliminovania vlastností. Výsledkom práce je prediktor obtiažnosti Rush Hour a identifikované významné vlastnosti, čo je možné v praxi použiť na automatické generovanie primerane obtiažnych inštancií problému.

Kľúčové slová: analýza stavového grafu, predpovedanie času, Rush Hour, významné vlastnosti, strojové učenie, zber dát

Abstract

The goal of this thesis is analyzing the way people solve specific computationally hard logic puzzles and analyzing how syntactic features of a problem instance affect the difficulty of said instance. The solution to this problem has a basis in the analysis of state space of problem instance. In this thesis, we investigate logic puzzle Rush Hour, which belongs to the set of PSPACE-complete problems. We created a website for collecting data from solving the problem by real players. For the purpose of analysis, we created a program for drawing 3D graphs of state space. The program also draws graphs weighted by collected data, the visits and time spent in edges and vertices of a graph. We used machine learning on data collected from the website along with external data source for predicting the time required to solve the problem. To simplify the prediction of problem difficulty, we chose only some state space features. With the grid search method, we selected the best learning models and then extracted the best features by recursive feature elimination. The result of the thesis is a predictor of Rush Hour difficulty along with identified significant features of state space, which can be used on automatic generation of Rush Hour instances of appropriate difficulty.

Keywords: state space analysis, time prediction, Rush Hour, significant features, data collection

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Cieľom tejto práce bolo preskúmať akým spôsobom ľudia riešia ťažké logické úlohy. Ľudia od nepamäti riešia logické úlohy vo forme hlavolamov. Hlavolam je problém, hádanka, záhada, ktorá skúša vynaliezavosť jej riešiteľa. Všeobecne najznámejšou formou hlavolamov sú mechanické hlavolamy, ale môžu byť za ne považované aj iné úlohy ako krížovky, sudoku, rôzne formy doplňovačiek, osemsmeroviek a podobne. Rovnako môže byť za hlavolam považovaná šifra, hádanka či matematická slovná úloha. Určitou formou hlavolamu môžu byť aj úlohy niektorých logických hier, ako sú šachy či GO.

Riešenie takýchto úloh je prepojené aj s riešením zložitých logických, prípadne matematických problémov. Skúmanie vedie k poznaniu, preto naša práca mala za cieľ nájsť odpoveď, či je možné nájsť vlastnosti logickej úlohy, ktoré významne ovplyvňujú čas riešenia úlohy. Následne by bolo možné predpovedať obtiažnosť riešenia úlohy pre človeka a čas potrebný na jej vyriešenie. Predpovedanie týchto faktorov má veľký význam nielen pri zostavovaní logických hlavolamov a hier v zábavnom priemysle ale aj v oblasti vzdelávania. Vyučujúci by vedel pripraviť podľa predchádzajúcich výsledkov skupiny študentov také príklady, ktoré by boli podľa potreby rôzne časovo náročné. Neodradili by tak študentov prílišnou zložitosťou a neboli by ani príliš ľahké.

Významnou časťou práce bola analýza dostatočne veľkej vzorky reálnych dát o riešení logickej úlohy. Takéto množstvo dát v požadovanej štruktúre bolo veľmi ťažké získať, preto sme vytvorili portál s hrou za účelom zberu dát z riešenia logickej úlohy. Všetky hráčske pokusy na tomto portáli boli zaznamenávané do databázy a následne analyzované pomocou strojového učenia. Riešiteľov úloh sme oslovili pomocou sociálnych sietí a mailom so žiadosťou o hranie logickej hry Rush Hour, ktorú sme sami implementovali a sprístupnili cez portál.

Práca je členená do piatich kapitol a niekoľkých podkapitol. V prvej kapitole všeobecne popisujeme logickú úlohu Rush Hour. Pojednávame o jej zložitosti, a to ako výpočtovej, tak aj zložitosti pre ľudských riešiteľov. Taktiež v tejto kapitole uvádzame formálnu definíciu problému. V druhej kapitole popisujeme postup a spôsob návrhu našej implementácie online hry Rush Hour, technológiu, štruktúru a úrovne hry. Súčasťou je tiež štruktúra zozbieraných a externých dát formou jednoduchých štatistík. V tretej kapitole sa venujeme analýze inštancií problému a jej vizualizácii. V tejto kapitole sú zobrazené aj výstupné grafy jednotlivých zaujímavých levelov. V štvrtej kapitole

potom popisujeme použité metódy analýzy dát. Venujeme sa optimalizácii metód a nastaveniu jednotlivých parametrov. Obsahom piatej kapitoly sú výsledky skúmania. V závere uvádzame zistenia a výsledky, ku ktorým sme dospeli a odporúčanie pre ich využitie v budúcnosti.

Prvým výstupom práce je portál, ktorý aj naďalej môže slúžiť na zber dát. Portál je možné rozšíriť o ďalšie úrovne, prípadne inak vylepšiť a prilákať tak viac hráčov. Zozbierané dáta môžu v budúcnosti slúžiť na iné analýzy. Dôležitou časťou práce je popis správania ľudských riešiteľov pri hraní na jednotlivých úrovniach hry vo forme stavových grafov. Najdôležitejším výstupom našej práce je identifikovanie významných vlastností, ktoré najviac ovplyvňujú čas riešenia úlohy ľudským riešiteľom a zistenie s akou presnosťou na ich základe je možné tento čas predpovedať.

Kapitola 1

Zložitosť logických úloh a Rush Hour

História hlavolamov siaha až do Starého Egypta, Číny, Japonska či na Arabský polostrov. Už v roku 1893 vydal profesor Hoffman v Londýne knihu Hlavolamy staré a nové (Puzzles old and new) [13], v ktorej popisuje viac než 40 hlavolamov. Po prvýkrát sa pokúsil ustanoviť základnú kategorizáciu hlavolamov a ako hlavný problém tejto úlohy popisuje fakt, že prakticky nie je možné určiť jednoznačné kategórie tak, aby sa niektoré neprekrývali. V súčasnosti sa počet všeobecne uznávaných kategórií hlavolamov ustálil na desať (napríklad skladacie hlavolamy, miznúce hlavolamy a podobne) s niekoľkými podkategóriami.

Hlavolamy sú mnohokrát vytvárané ako forma zábavy, často však vychádzajú z vážnych matematických či logických problémov. V takýchto prípadoch môže byť skúmanie spôsobu ich riešenie významným príspevkom k výskumu vo viacerých vedných odboroch. Práve týmto druhom hlavolamov sme sa zaoberali v našej práci.

V úvode kapitoly sme uviedli výskumy, ktoré sa zoberajú problematikou obtiažnosti riešenia logických úloh človekom. Zároveň sme uviedli našu motiváciu pri výbere skúmaného hlavolamu, ktorý z jednoduchých pravidiel tvorí výpočtovo veľmi náročný problém. Zaujalo nás, ako človek napriek tomu dokáže takýto typ logickej úlohy riešiť. „Záhady nestrácajú svoje čaro, keď sú vyriešené, práve naopak, riešenie je často krajšie ako hlavolam.“ [24]

Ďalšie časti kapitoly popisujú nami zvolenú logickú hádanku, jej formálnu definíciu a bližšie rozoberajú pomocou simulácie iného problému, prečo je táto hádanka tak výpočtovo náročná.

1.1 Riešenie logických úloh človekom

Jednou z hlavných otázok, ktoré vznikajú pri riešení matematických, logických úloh alebo logických hádaniek a puzzle, je ako ťažké je pre ľudí nájsť riešenie. Zaujímavé je nielen to, či ľudia vedia vyriešiť daný problém, ale aj to, ako dlho im to bude trvať. Ak riešenie úlohy trvá príliš dlhý alebo príliš krátky čas, môže to byť pre riešiteľa demotivujúce.

Už v minulosti bolo podrobené skúmaniu ako ľudia riešia logické úlohy, a to nielen vo vedných odboroch ako je psychológia ale aj v informatike. Napríklad v roku 2010 Ashlock a Schonfeld prezentovali automatické hodnotenie obtiažnosti Sokobanu [3]. Metódu automatického hodnotenia obtiažnosti úrovní logických hier popisujú aj Kreveld, Löffler a Mutser [22]. Prácu zamerali na tri hry - Flow, Lazors a Move. Ich metóda využíva viacero atribútov hier, ktoré kombinuje do funkcie obtiažnosti. Bola popísaná aj problematika obtiažnosti logickej hry Nurikabe [4], Tilt Maze a Sokoban [21] a Rush Hour [5]. Pre nás zaujímavý bol výskum tímu z Brna. Jarušek a Pelánek vydali viacero odborných článkov, napríklad analyzujú spôsob hrania hry Sokoban človekom [18] alebo obtiažnosť úrovní hier Sokoban a Rush Hour [17, 15].

Pre účely našej práce sme zvažovali skúmanie obtiažnosti problémov pre hry Sokoban, Rush Hour alebo Lemmings. Hru Sokoban sme vylúčili z dôvodu, že sa obtiažnosťou tejto hry zaoberali viaceré výskumy a Lemmings patrí iba do triedy NP-úplných problémov, zatiaľ čo Rush Hour patrí do vyššej triedy PSPACE-úplných problémov. Vybrali sme Rush Hour, ktorý je výpočtovo ťažší a preto bol jeho spôsob riešenia ľudskými riešiteľmi zaujímavejší, naviac pre potrebu zberu dát je implementácia Rush Hour jednoduchšia.

1.2 Rush Hour

Rush Hour (Obr. 1.1) je známy logický hlavolam, ktorý vynašiel v sedemdesiatych rokoch dvadsiateho storočia Nob Yoshigahara. Väčšinou sa hrá na ploche 6×6, ale existujú aj relaxované verzie, kde plocha môže byť ľubovoľná.

Vo východiskovom stave sú na ploche rozmiestnené autá, reprezentované obdĺžnikmi veľkostí 1×2 a 1×3, z ktorých jedno je označené ako cieľové. Tiež na jednej strane plochy je vyznačený východ. Cieľom hry je dostať cieľové auto mimo hraciu plochu, a tým prejsť cez upchatú cestu.

V jednom kroku riešenia je možné pohnúť jedným autom o ľubovoľný počet políčok dopredu alebo dozadu za podmienky, že sa nikdy nemôže prekryť so žiadnym iným autom a žiadnou svojou časťou opustiť hraciu plochu, okrem špeciálneho prípadu cieľového auta.

Obr. 1.1: Klasická verzia Rush Hour [1]

1.2.1 Formálna definícia

Zadefinovali sme si *relaxovanú* Rush Hour formálne. Prvá definícia, ktorú sme potrebovali bola, ako vyzerá jedna *inštancia* (zadanie) Rush Hour.

Definícia 1.2.1 (Zadanie relaxovanej Rush Hour) *je 6-tica* (w, h, N, C, V, H) *kde* $w, h \in \mathbb{Z}$ *sú rozmery hracej plochy,* $N \in \mathbb{Z}$ *je počet vstupných blokov,* C *je postupnosť blokov* $(c_i)_{i=0}^{N}$*, kde* $c_i = (x_i, y_i, w_i, h_i) \in \mathbb{Z}^4$ *a* $V, H \subseteq \mathbb{Z}_N$ *sú množiny vertikálnych a horizontálnych blokov. Musí platiť* $V \cap H = \emptyset$ *a* $V \cup H = \mathbb{Z}_N$*, teda že každý blok je práve v jednej z týchto množín.*

Zadefinovali sme všetky objekty, s ktorými pracujeme - hraciu plochu, *bloky* (autá), ich rozmery a počiatočné umiestnenia. Pri *nerelaxovanej* Rush Hour existuje naviac niekoľko ďalších podmienok:

Definícia 1.2.2 (Zadanie nerelaxovanej Rush Hour) *je také zadanie relaxovanej Rush Hour, ktoré navyše spĺňa tieto podmienky:*

- $w = h = 6$
- $\forall c_i \in C, c_i = (x_i, y_i, w_i, h_i) : i \in V \implies h_i = 1 \wedge w_i \in \{2, 3\}$
- $\forall c_i \in C, c_i = (x_i, y_i, w_i, h_i) : i \in H \implies w_i = 1 \wedge h_i \in \{2, 3\}$

Prvá podmienka hovorí o obmedzení plochy na 6×6, pričom druhá a tretia obmedzuje všetky bloky na rozmery 1×2 alebo 1×3 v smere, v ktorom sa môžu hýbať.

Definícia 1.2.3 (Platná konfigurácia) *je postupnosť* $K = (k_i)_{i=0}^{N}, k_i \in \mathbb{Z}$*, kde* k_i *je posun* i*-teho bloku od počiatočného stavu. Zadefinujeme si*

$$B(a) = \{(x_a + i, y_a + j) \mid 0 \leq i < w_a \wedge 0 \leq j < h_a\}$$

ako funkciu obsadených pozícií a*-tým blokom.*

Potom pre všetky $i \in \mathbb{Z}_N$ *musí platiť:*

- $i \in H \implies 0 \leq x_i + k_i \leq w - w_i$
- $i \in V \implies 0 \leq y_i + k_i \leq h - h_i$
- $\forall j \in \mathbb{Z}_N, j \neq i : B(i) \cap B(j) = \emptyset$

a navyše konfigurácia $K^0 = (0)_{i=0}^N$ *musí byť platná a nazývame ju* ***počiatočná konfigurácia****.*

Prvé dve podmienky hovoria o tom, že žiadne auto nemôže vyjsť z plochy, tretia podmienka hovorí o tom, že žiadne dve autá sa nesmú prekrývať. *Počiatočná konfigurácia* je špeciálna konfigurácia, ktorá navyše obmedzuje aké môže byť zadanie. Ak pre zadanie počiatočná konfigurácia nie je platná, tak toto zadanie nie je validné.

Ďalej sme si zadefinovali vzťah medzi konfiguráciami - reláciu ťahu.

Definícia 1.2.4 (Jednoduchý ťah) *je relácia* $\vdash$ *medzi dvoma konfiguráciami* K^p *a* K^n*. Platí* $K^p \vdash K^n$ *práve vtedy, keď* $\exists i \in \mathbb{Z}_N$ *také, že sú splnené všetky nasledujúce podmienky:*

- K^p, K^n *sú platné konfigurácie*
- $k_i^n - k_i^p \in \{-1, 1\}$
- $\forall j \in \mathbb{Z}_N, j \neq i : k_j^n = k_j^p$

a takýto ťah značíme $K^p \vdash_{i+1} K^n$ *respektíve* $K^p \vdash_{i-1} K^n$ *podľa numerickej hodnoty* $k_i^n - k_i^p$*.*

Tranzitívny uzáver jednoduchého ťahu označíme $\vdash^*$ *a budeme hovoriť, že konfigurácia* K^B ***je dosiahnuteľná*** *z* K^A *práve vtedy keď platí* $K^A \vdash^* K^B$*.*

Jednoduchý ťah je teda zmena jednej časti konfiguračnej postupnosti o 1. Definovali sme jednoduchý ťah, nie zložený, aj keď v originálnom zadaní sú povolené zložené ťahy, lebo pri jednoduchých ťahoch stačí, že počiatočná a koncová konfigurácia je platná. Zložený ťah sme zadefinovali pomocou jednoduchého:

Definícia 1.2.5 (Zložený ťah) *je relácia* $\vdash^Z$ *medzi dvoma konfiguráciami* K^p *a* K^n*. Platí* $K^p \vdash^Z K^n$ *práve vtedy, keď* $\exists i \in \mathbb{Z}_N$ *a* $\exists K_0 ... K_k$ *také, že sú splnené všetky nasledujúce podmienky:*

- $K_0 = K^p$ *a* $K_k = K^n$
- *všetky* $K_0, ... K_k$ *sú platné konfigurácie*
- *pre všetky* $0 \leq j < k$ *platí* $K_j \vdash_{i+1} K_{j+1}$

a takýto ťah značíme $K^p \vdash^Z_{i+k} K^n$. *Ťah* $K^p \vdash^Z_{i-k} K^n$ *značíme taký ťah, ktorý spĺňa všetky horeuvedené podmienky ale pre reláciu* $\vdash_{i-1}$.

Zložený ťah je pri Rush Hour považovaný za jednotkový ťah, teda počet ťahov potrebných na riešenie je práve počet potrebných zložených ťahov. Dá sa zadefinovať ako k-opakovaní rovnakého jednoduchého ťahu.

Nakoniec sme zadefinovali cieľ hry:

Definícia 1.2.6 (Riešenie Rush Hour) *Majme zadanie* H, *počiatočnú konfiguráciu* K^0 *a konfiguráciu* K^e, *pre ktorú platí*

$$(0 \in H \implies k^e_0 + x_0 = w - w_0) \wedge (0 \in V \implies k^e_0 + y_0 = h - h_0)$$

Takúto konfiguráciu nazývame koncovou.

Ak platí $K^s \vdash^* K^e$, *tak potom* K^e *je víťaznou konfiguráciou, a postupnosť jednoduchých alebo zložených ťahov* (t_i), *ktorou bola vyrobená* K^e *z* K^s *voláme riešením Rush Hour.*

Inak povedané, koncová konfigurácia je taká, kde blok c_0 je na pravom alebo dolnom okraji plochy, podľa toho, ktorým smerom sa vie pohybovať. Teda toto cieľové auto je bezprostredne pri východe z plochy a nemôže mu nič zabrániť vyjsť. Oproti normálnej definícií (auto musí vyjsť z plochy) sa minimálny počet zložených ťahov nemení, keďže posledný ťah pred prvou výhernou konfiguráciou sa musel ťahať cieľovým autom a tento ťah sa dá jednoducho predĺžiť mimo plochu.

1.2.2 Príklad zadania

V tejto sekcii sme uviedli príklad formálneho zadania Rush Hour na konkrétnom prípade úrovne 176 Crilade. Keďže všetky úrovne boli nerelaxovaná Rush Hour, tak sme vedeli že $w = h = 6$. Mali sme $N = 6$ rôznych blokov, pre ktoré sme vedeli zadefinovať ich rozmery nasledovne:

$$
\begin{aligned}
C_c = (\\
&(0,2,2,1),\\
&(1,3,3,1),\\
&(4,4,2,1),\\
&(2,0,1,3),\\
&(3,1,1,2),\\
&(3,4,1,2),\\
&(4,0,1,2),\\
&(4,2,1,2)\\
)
\end{aligned}
$$

Kde podľa definície 1.2.1 každá štvorica má význam (x, y, šírka, výška). Posledné dve časti zadania boli množiny vertikálnych a horizontálnych áut $V_c = \{3,4,5,6,7\}$ a $H_c = \{0,1,2\}$.

Potom $Crilade = (6,6,8,C_c,V_c,H_c)$ je platné zadanie Rush Hour.

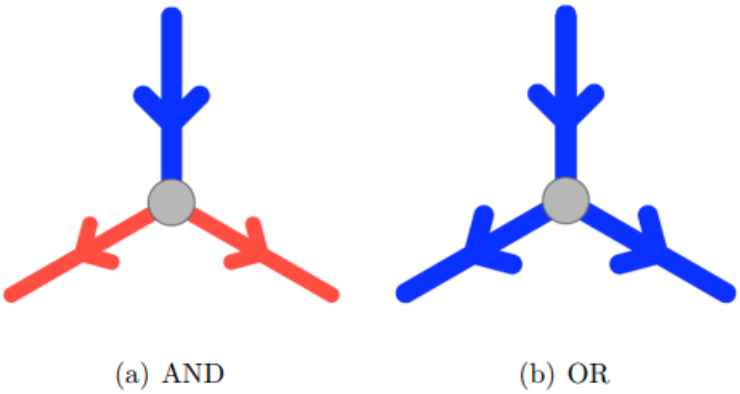
1.2.3 Výpočtová zložitosť

Výpočtovou zložitosťou sa zaoberá teória zložitosti, ktorá definuje rôzne *triedy výpočtovej zložitosti* problémov. Výpočtová zložitosť často nesúvisí so zložitosťou riešenia ľudským riešiteľom, aj keď niektoré triedy zložitosti sú ľuďmi ľahko riešiteľné. Preto sme sa bližšie pozreli na výpočtovú zložitosť Rush Hour.

Problém rozhodnúť či daná inštancia Rush Hour je splniteľná patrí do triedy PSPACE-úplných problémov. Existuje niekoľko rôznych dôkazov tohto tvrdenia [27, 7]. My sme sa zaoberali dôkazom [27], ktorý popisuje redukciu na iný PSPACE-úplný problém nazývaný Nedeterministická obmedzujúca logika (Nondeterministic constraint logic), skratkou NCL.

Nedeterministická obmedzujúca logika

Táto hra sa hrá na *orientovanom grafe* $G = (V,E)$, ktorý definuje pravidlá. Každá hrana je orientovaná a jej váha je jedna z hodnôt $\{1,2\}$. Každý vrchol má maximálny stupeň 3, to znamená, že môže mať maximálne 3 hrany. *Prítok* vrcholu sa definuje ako súčet váh všetkých hrán, ktoré sú orientované smerom do vrcholu. Každý vrchol grafu má zadefinovaný minimálny prítok, ktorý musí spĺňať. Konfigurácia grafu je *platná* práve vtedy, keď prítok každého vrcholu je väčší alebo rovný jeho minimálnemu prítoku.



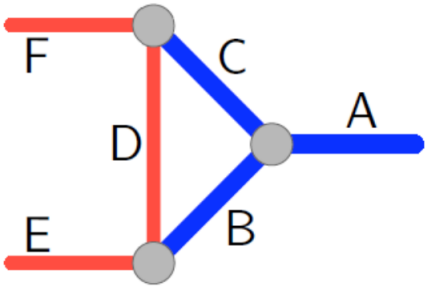
(a) AND (b) OR

Obr. 1.2: NCL útvary [27]

Platný ťah v tejto hre je otočenie orientácií jednej hrany tak, aby znova vznikla platná konfigurácia, teda aby boli splnené všetky obmedzenia vo vrcholoch. Za výhru NCL sa považuje otočenie danej hrany v grafe.

Je dokázané [10], že NCL je PSPACE-úplná, ak je graf *planárny* a poskladaný iba z útvarov zobrazených na obrázku 1.2, kde červené hrany majú hodnotu 1, a modré majú hodnotu 2. Oba vrcholy majú minimálny prítok 2. Tieto útvary sa nazývajú OR a AND vrcholmi, keďže svojou funkciou pripomínajú logické hradlá.

Aby sme zjednodušili prácu, tak sme pridali ešte dva typy útvarov, a to jednoduchú linku a protected OR (chránený OR). Jednoduchá linka je len vrchol stupňa 2, ktorý má minimálny prítok 1. Tento útvar iba predlžuje hranu alebo dokáže vyrobiť hranu s váhou 1 na jednej strane a váhou 2 na druhej strane. Protected OR (Obr. 1.3) je útvar, ktorý sa správa ako OR, až na to, že dve „vstupné“ hrany nemôžu byť naraz orientované dovnútra.



Obr. 1.3: NCL protected OR [27]

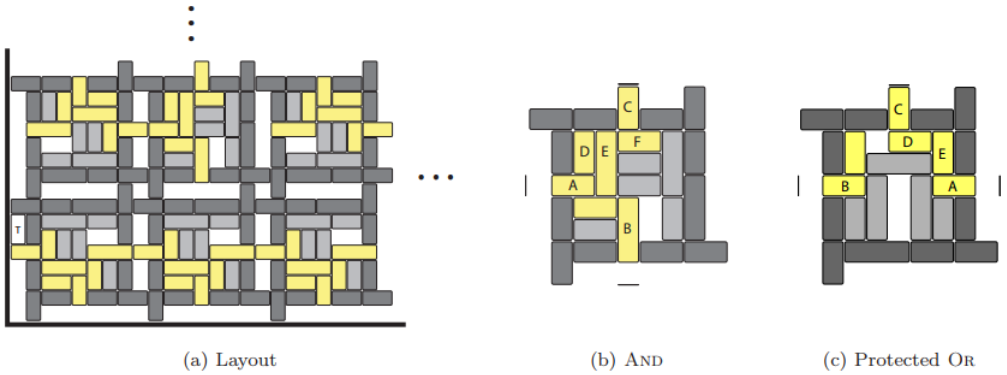
Keďže s pomocou AND, protected OR a jednoduchých liniek vieme vyrobiť planárny podgraf, ktorý sa správa ako OR [10], tak NCL iba s týmito 3 prvkami je stále PSPACE-

úplný.

Redukcia NCL na Rush Hour

Aby sa dalo zredukovať takéto NCL na Rush Hour, musí sa vedieť pre každý NCL problém skonštruovať ekvivalentný problém Rush Hour.

Je možné skonštruovať jednoduché štvorcové usporiadania blokov v Rush Hour, ktoré majú rovnakú funkciu ako útvary NCL. Pri týchto útvaroch sú niektoré bloky vystupujúce zo štvorca, ktoré reprezentujú hrany. To či sú vysunuté alebo nie, reprezentuje orientáciu danej hrany. Tieto konštrukcie sú znázornené na obrázku 1.4. Útvar, ktorý zodpovedá jednoduchej linke je konštruovateľný podobne.



(a) Layout (b) AND (c) Protected OR

Obr. 1.4: Konštrukcia AND a OR brán v Rush Hour [27]

Keďže grafy problému NCL sú planárne, vieme útvary usporiadať do plochy tak, aby sme ich vedeli vzájomne prepojiť. Na obrázku 1.4 je možné vidieť rôzne zafarbené bloky, každý z nich predstavuje jeden blok Rush Hour, sivou farbou sú vyznačené nehybné bloky. Tmavo sivé bloky tvoria základ štruktúry, a tým že sa opierajú o okraje plochy na všetkých stranách nám zaručujú, že sa nepohnú. Slabo sivé bloky sú navrhnuté tak, aby sa nepohli v rámci jedného útvaru, alebo v prípade ich pohybu neovplyvnili možnosti pohybu žltých blokov.

Cieľ v tejto inštancii je dostať biely blok T (na obrázku 1.4 vľavo dole) von z plochy, čo je ťah zodpovedajúci otočeniu cieľovej hrany v NCL.

PSPACE-úplný

Pomocou redukcie NCL sa vie dokázať, že Rush Hour je PSPACE-*ťažký* problém. Na to, aby sa dokázalo že je PSPACE-úplný, stačí zostrojiť PSPACE program, ktorý ho rieši.

Takýto program skonštruovať je jednoduché. V polynomiálnej pamäti sa zostrojí stav hracej plochy Rush Hour a nedeterministicky sa spraví ťah z tejto pozície, pričom

je zapamätaný len aktuálny stav hracej plochy. Nedeterminizmus zaručí že sa budú robiť iba správne ťahy, a teda tento program rieši Rush Hour a je v triede NSPACE. Navyše zo Savitchovej Teorémy [29] sa vie že PSPACE = NSPACE, a teda tento program je aj PSPACE.

Z uvedeného vyplýva že Rush Hour je PSPACE-úplný problém.

Kapitola 2

Web portál na zber dát

Na zber dát sme vytvorili jednoduchý web portál s implementáciou hry, ktorá obsahuje dvesto úrovní a rebríček na motivovanie hráčov. Tento portál je prístupný na adrese `<https://rushhourpuzzle.pythonanywhere.com/>`. Hráčov sme oslovili prostredníctvom sociálnych sietí a mail listov. Hranie hry prebehlo v niekoľkých vlnách, podľa času postupného zverejňovania. Zaujímavé bolo práve to, že hráči, ktorí hrali úrovne podľa poradia (stupňa obťažnosti) postupne hru vzdali, keď sa dostali po úrovne, ktoré boli už časovo náročnejšie. V konečnom poradí je však aj niekoľko hráčov, ktorí vyriešili všetky úrovne. Web portál sme sprevádzkovali pred viac ako pol rokom z dôvodu vytvorenia potrebného času pre zber dostatočného množstva dát na analýzu.

2.1 Použité technológie

Portál je založený na Python frameworku Django [2]. Framework sme zvolili, nakoľko umožňuje veľmi rýchly vývoj webových aplikácií a už priamo obsahuje niektoré dôležité moduly väčšiny webových aplikácií. Automaticky sprístupňuje databázu, stránky pre administrátorov a tiež prihlasovaciu obrazovku, ktorú sme do našej aplikácie implementovali hlavne za účelom zverejňovania výsledkov, ktoré slúžia na motiváciu hráčov. Na druhej strane framework nezabraňuje škálovateľnosti aplikácie a podporuje jej dobrú udržiavateľnosť. Naviac Django ako aj Python aktuálne obsahuje veľké množstvo knižníc, ktoré uľahčujú a urýchľujú prácu vývojára a väčšina je vyvíjaná ako open-source.

Hlavné výhody použitia Django frameworku sú:

- je založený na programovacom jazyku Python,
- filozofia *Batteries included* - obsahuje mnoho modulov (auth, admin, redirect, session...),
- Model-View-Controller (skratkou MVC) dizajnový vzor,
- spoľahlivá prevádzka viac ako 12 rokov,

- veľké množstvo knižníc,
- veľká komunita,
- a hlavne kvalitná dokumentácia.

Ako úložisko sme použili databázu SQLite [12]. Táto databáza je veľmi obľúbená pre podobný typ projektov a to z nasledovných dôvodov:

- nie je potrebná žiadna inštalácia,
- môže pracovať bez servera,
- je to jeden súbor na úložisku,
- je multiplatformová,
- je kompaktná,
- má čitateľný kód,
- má premenlivú dĺžku záznamov,
- podporuje manifest typovanie,
- je open-source.

Uvedený typ databázy podporuje aj hosting, na ktorom prevádzkujeme portál. V účte typu začiatočník (beginner) je obsiahnutý úložný priestor 521 MB, čo bolo pre potreby nášho projektu postačujúce. V prípade, že by hráčov bolo viac, čo sa však nestalo, kopírovaním záznamov z databázy na lokálny disk by sme postupne uvoľňovali miesto pre nové údaje. Nastavenie projektu na tomto hostingu bolo veľmi jednoduché a rýchle a vývoj aplikácie sme začali v priebehu pár minút.

Výber hostingu sme podmienili hlavne:

- podporou Django a Python,
- podporou SQLite,
- spoľahlivou prevádzkou s podporou,
- rýchlou inštaláciou potrebných modulov,
- obsahom veľkého množstva Python knižníc,
- možnosťou jednoduchého a rýchleho nasadenia,
- cenou (nie je spoplatnený).

Na základe týchto podmienok pripadali pre náš projekt do úvahy hlavne PythonAnywhere [30] a Heroku [28], z ktorých sme si zvolili PythonAnywhere najmä kvôli mohutnejšej podpore Python.

Pre animáciu hry sme použili klientsky Javascript, ktorý zároveň kontroluje pravidlá hry, či sú splnené a posiela na server údaje o postupoch hráčov pri hre. Použitý Javascript využíva na kreslenie framework Paper.js [23], ktorý je kvalitne navrhnutý a poskytuje mnoho funkcií pre prácu s vektorovou a bitmapovou grafikou, a tiež ponúka možnosť usporiadania grafických objektov do Document Object Modelu. Obsahuje aj mnoho matematických funkcií, ktoré sme použili pri programovaní portálu.

Kritériá pre výber javascript frameworku sme si stanovili nasledovne:

- podpora práce s grafikou a tvarmi (shapes),
- schopnosť vyhodnocovať kolízie dvoch objektov,
- nie príliš zbytočná robustnosť frameworku,
- jednoduchá a rýchla implementácia.

Práve z dôvodu robustnosti sme hneď na začiatku vylúčili Angular a React, ktoré sa aktuálne najviac používajú na tvorbu moderných webových aplikácií. Hľadali sme teda odľahčenú verziu pre prácu s grafikou a nakoniec sme sa rozhodovali medzi Fabric.js [31], HTML Canvas Library [19] a uvádzaným Paper.js [23], ktorý sme si vybrali práve pre jeho podporu vektorovej grafiky, konzistentnosť a vynikajúcu architektúru.

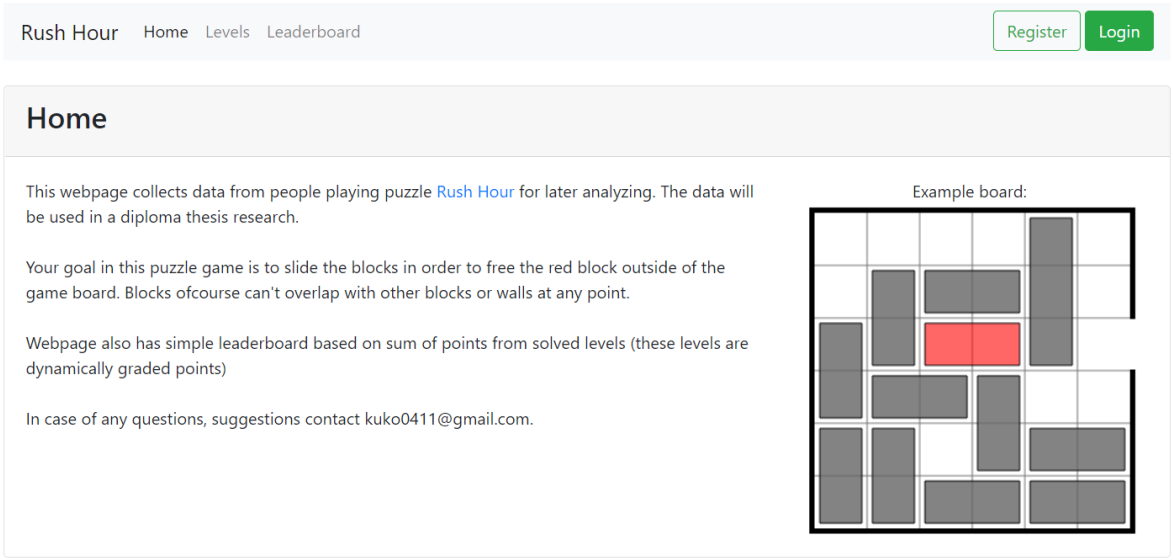
2.2 Štruktúra portálu

Portál obsahuje jednoduchú titulnú stránku (Obr. 2.1), ktorá vysvetľuje účel portálu a stručné pravidlá hry. Uvádza aj mailový kontakt na vývojára.

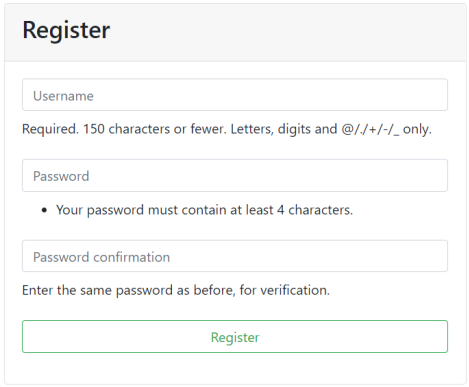
Pre účely výsledkovej listiny sme potrebovali identifikovať používateľa. Preto sme vytvorili stránku na registráciu a prihlásenie pomocou prezývky a hesla, ktoré sa ukladá v hash forme. Ukladá sa iba priebeh hry hráča a keďže všetky ukladané údaje nie sú pre používateľa citlivé, tak nie sú v databáze kryptované. Pre vytvorenie používateľského konta bolo nutné vytvoriť aj obrazovku pre registráciu (Obr. 2.2a), kde sa uvádzajú požiadavky na prihlasovacie meno a heslo. Obrazovka pre prihlásenie je zobrazená na obrázku 2.2b.

Ďalšia stránka portálu obsahuje zoznam úrovní, ktorý je utriedený podľa počtu bodov, ktoré používateľ získa za ich riešenie (Obr. 2.3). V zozname je uvedený aj minimálny počet ťahov, počet konfigurácií a počet áut v hracom pláne.

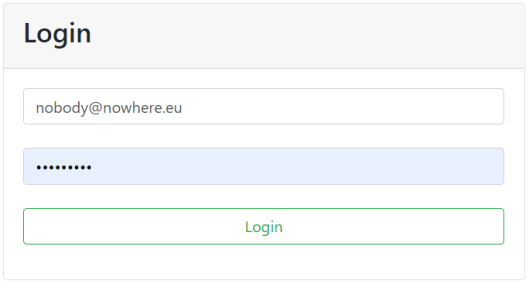
Stránky pre jednotlivé úrovne (Obr. 2.4) obsahujú hrací plán a na motivovanie hráčov zobrazuje jednoduchú výsledkovú listinu hráčov s ich najlepšími riešeniami, buď



Obr. 2.1: Web portál



(a) Registrácia používateľa



(b) Prihlásenie používateľa

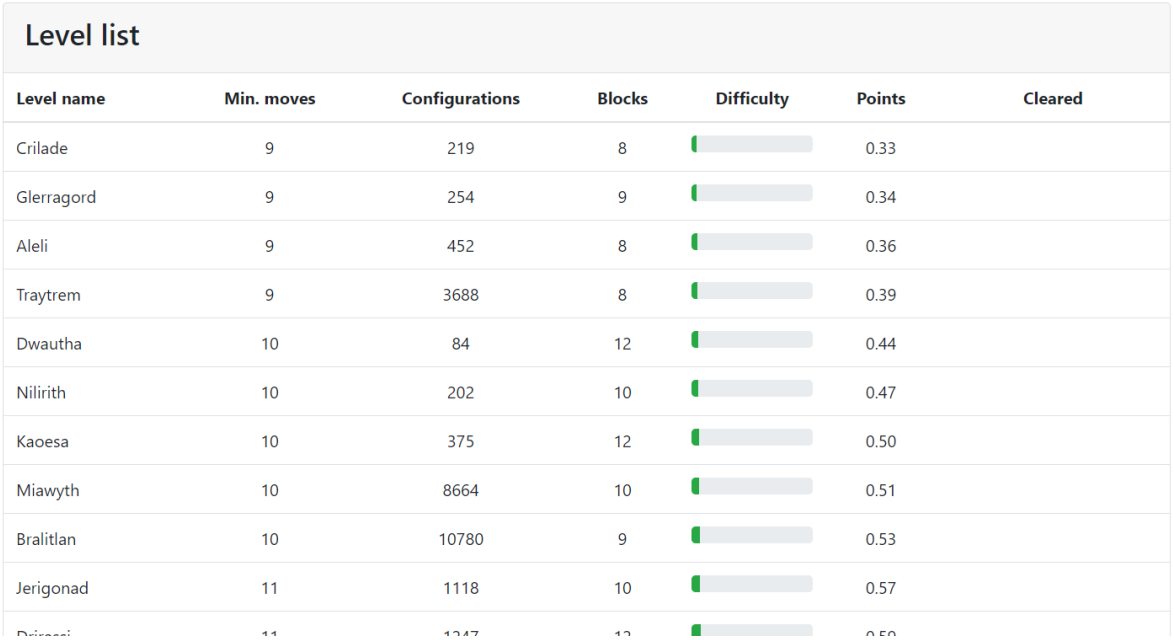
Obr. 2.2: Prihlásenie a registrácia

podľa počtu ťahov alebo rýchlosti riešenia. Pohybovať autami je možné potiahnutím myšou v smere pohybu auta. Aplikácia vyhodnocuje kolíziu a nedovolí potiahnuť autom za hranu hracieho plánu alebo cez druhé vozidlo. Cieľové auto je odlíšené farebne a východ z hracieho plánu je určený výsekom v orámovaní hracieho plánu.

2.3 Zdroj zadaní úrovní

Zadania úrovní sme použili z databázy Rush Hour úrovní [25], ktorá obsahuje „zaujímavé“ úrovne Rush Hour. Podľa autora úrovní zaujímavé problémy sú také, ktoré spĺňajú nasledovné podmienky:

- žiadny riadok ani stĺpec nemôže byť plný blokov, ktoré sa nemôžu pohnúť, keďže to len zmenšuje plochu,



Level list

Level name	Min. moves	Configurations	Blocks	Difficulty	Points	Cleared
Crilade	9	219	8		0.33	
Glerragord	9	254	9		0.34	
Aleli	9	452	8		0.36	
Traytrem	9	3688	8		0.39	
Dwautha	10	84	12		0.44	
Nilirith	10	202	10		0.47	
Kaoesa	10	375	12		0.50	
Miawyth	10	8664	10		0.51	
Bralitlan	10	10780	9		0.53	
Jerigonad	11	1118	10		0.57	

Obr. 2.3: Zoznam levelov

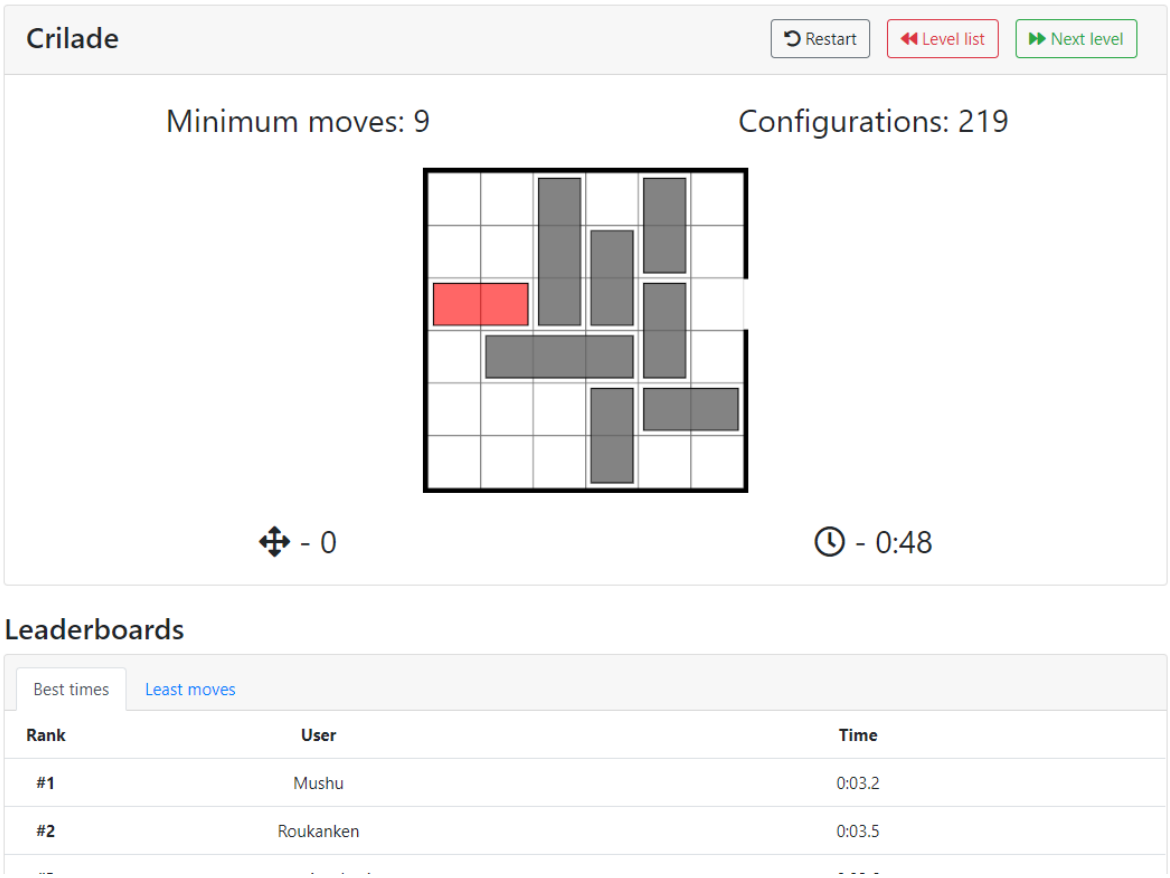
- nemôžu sa vyskytnúť dva rôzne bloky, ktoré môžu vyjsť z plochy,
- žiadne dve úrovne v databáze nie sú navzájom variáciami, pohybmi blokov sa nedá transformovať jedna úroveň na druhú,
- úrovne sú *úplne nevyriešené* - pohybom blokov nie je možné zvýšiť počet ťahov potrebných na vyriešenie,
- úrovne sú *minimálne* - každý blok je dôležitý, a ak ktorýkoľvek odstránime, zmení to riešenie.

2.4 Zbierané dáta

Zozbierané dáta boli extrahované z databázy a sú dostupné vo forme `csv` na priloženom CD.

Nakoľko serverový Python kód nekontroloval, či pravidlá hry boli splnené, bolo tak možné manipulovať REST API, aby akceptovalo aj neplatné riešenia. Napríklad sa vyskytol taký prípad, keď používateľ „vyriešil“ úlohu na nula ťahov, ale tohto hráča sme odstránili už počas zberu dát.

Aby všetky pravidlá hry boli dodržané, databázu riešení sme po stiahnutí skontrolovali. Na uvedený účel sme vytvorili jednoduchý Python skript, ktorý hru odsimuluje, skontroluje výsledok a platnosť všetkých ťahov. Okrem spomenutého riešiteľa, všetky ostatné riešenia boli platné.



Obr. 2.4: Stránka hracej plochy

2.4.1 Štruktúra dát

Pre každé riešenie, alebo pokus o riešenie je v databáze stránky vytváraný záznam, vygenerovaný javascriptom. Tento záznam obsahuje všetky ťahy, ktoré používateľ urobil a každý ťah obsahuje nasledovné údaje:

- **id** - identifikačné číslo auta, ktorým bolo pohnuté,
- **start** - čas začatia ťahu v milisekundách,
- **end** - čas ukončenia ťahu v milisekundách,
- **pos** - koncová pozícia ťahu, hodnota nezafixovanej súradnice ľavého horného rohu,
- **min** a **max** - najmenšia a najväčšia súradnica, na ktorú sa auto pohlo počas ťahania.

Samozrejme, každé riešenie je priradené k úrovni a používateľovi, ktorý ho riešil. Výsledná štruktúra údajov je v nasledovných základných tabuľkách:

- `auth_user` - tabuľka používateľov (generovaná frameworkom), obsahuje registrovaných používateľov,

- `puzzle_level` - tabuľka úrovní, obsahuje údaje jednotlivých úrovní,
- `puzzle_clear` - tabuľka riešení, obsahuje údaje o hraní jednotlivých úrovní konkrétnym hráčom.

Jednotlivé ťahy autami a rozloženie úrovní sú ukladané do databázy v JSON formáte v rámci uvedených tabuliek, keďže z hľadiska implementácie nebolo potrebné relačné uloženie údajov.

2.4.2 Štatistika dát

Počas prevádzky portálu (v rozsahu približne pol roka) sme zozbierali nasledujúce množstvo údajov:

Počet používateľov	107
Počet úrovní	200
Úspešné riešenia	3 699
Unikátne úspešné riešenia	2 834
Priemer používateľov na úroveň	15,2
Priemer úrovní na používateľa	23,0
Celkový čas riešení	20 dní a 6,5 hodín
Celkový čas úspešných riešení	4 dni a 14,2 hodín
Celkový počet ťahov	148 340

Tabuľka 2.1: Štatistika zozbieraných dát portálom

Ako vyplýva z údajov v tabuľke 2.3, počet hráčov nebol dostatočný, čo malo vplyv na kvalitu údajov. Niektoré úrovne vyriešil veľmi malý počet hráčov a údaje z nich neboli dostatočné pre ďalšie použitie. Napríklad polovica používateľov vyriešila len 13 úrovní. Všetky úrovne vyriešili len štyria používatelia. Na skúmanie ako ľudia pristupujú k riešeniu ťažkých logických úloh boli dáta postačujúce. Pre predpoveď obtiažnosti úrovne, prípadne času jej riešenia však dáta neboli postačujúce, aj napriek pomerne dlhému času prevádzky hry.

Preto sme zisťovali, či niekde ešte nie je prístupná sada údajov pre Rush Hour zozbieraná podobným spôsobom. Podarilo sa nám získať údaje priamo od autorov podobného web portálu UmímeMatiku [16]. Tento portál vychádza z predchádzajúceho už zrušeného portálu Tutor `<http://tutor.fi.muni.cz>` a pokračuje v modelovaní logických hádaniek typu Sokoban, Rush Hour a podobne. Autori do nového portálu doplnili zber dodatočných atribútov (napríklad detailný postup každého riešenia), ktoré

boli pre naše skúmanie nevyhnutné a pôvodný portál Tutor ich neobsahoval. Na stránkach portálu UmímeMatiku je možné riešiť úlohy logiky, matematiky, českého jazyka a iných oblastí.

Autori obsahu už dlhé roky skúmajú ako sa ľudia správajú pri riešení logických úloh a vydali množstvo publikácií. Údaje zo zrušeného portálu Tutor boli analyzované aj v iných prácach, napríklad pre hlavolam Sokoban [21]. Údaje na portáloch boli zbierané dlho a bolo ich dostatočné množstvo:

Počet používateľov	7 404
Počet úrovní	61
Úspešné riešenia	58 564
Unikátne úspešné riešenia	55 230
Priemer používateľov na úroveň	933,0
Celkový čas riešení	32 dní a 17,9 hodín
Celkový čas úspešných riešení	20 dni a 5,8 hodín
Celkový počet ťahov	1 056 934

Tabuľka 2.2: Štatistika zozbieraných dát UmímeMatiku

Aby sme mohli údaje z tohto dátového zdroja zlúčiť s našimi údajmi, bolo potrebné vyextrahovať údaje z oboch zdrojov. Následne bola potrebná transformácia časových jednotiek, nakoľko nami zozbierané údaje o čase boli v iných jednotkách ako v druhom zdroji dát. Výsledná množina údajov pre predikciu bola nasledovná:

Počet používateľov	7 513
Počet úrovní	261
Úspešné riešenia	62 263
Unikátne úspešné riešenia	58 064
Celkový čas riešení	53 dní a 0,4 hodín
Celkový čas úspešných riešení	24 dní a 20 hodín
Celkový počet ťahov	1 214 274

Tabuľka 2.3: Štatistika spojených dát

S takto získaným zdrojom dát sme pracovali na odhade času riešenia ako je popísané v kapitole 4.

Kapitola 3

Analýza inštancií problému

V tejto kapitole sa zaoberáme možnosťami analýzy zozbieraných údajov. Vyskúšali sme rôzne prístupy analýzy a snažili sme sa zistiť, či sme schopní zo zozbieraných dát identifikovať a popísať spoločné črty správania ľudských riešiteľov pri riešení logických úloh.

Väčšiu časť kapitoly tvorí skúmanie stavového priestoru, nakoľko hlavolam Rush Hour je jeden z PSPACE-úplných problémov, a tým je jeho zložitosť úzko previazaná so všetkými jeho možnými konfiguráciami. Tento stavový priestor sme dali do vzťahu s údajmi o riešiteľských návštevách a časoch pre jednotlivé úrovne hry a pomocou vizualizácie sme zobrazili zozbierané priebehy riešení.

3.1 Dekompozícia

Jedným z možných štýlov analýzy je dekompozícia, teda rozloženie problému na viacero podproblémov. V iných logických hlavolamoch bolo preukázané, že existuje korelácia medzi schopnosťou dekomponovať problém a priemerným časom, ktorý ľudia potrebujú na vyriešenie danej inštancie problému [15].

Konkrétne, bolo preukázané, že dva menšie podproblémy sú zväčša riešiteľné rýchlejšie ako jeden veľký problém.

Avšak problém Rush Hour je veľmi prepojený a veľmi málo jeho inštancií je dekomponovateľných, muselo by sa zabezpečiť významné oddelenie jednotlivých častí, napríklad nehybnou, alebo málo hybnou stenou z blokov. Na tradičnej ploche 6×6 priestor na takýto predel ale nie je, pretože všetky takéto inštancie problému by boli príliš ľahké alebo triviálne.

Pre väčšie plochy sa niektoré inštancie dajú dekomponovať, napríklad konfigurácia použitá pri dôkaze PSPACE-úplnosti v kapitole 1.2.3 je príkladom jednoducho dekomponovateľného problému, a aj väčšie inštancie skonštruované týmto spôsobom by boli relatívne ľahko riešiteľné človekom v porovnaní s počtom blokov a veľkosťou plochy

ktorú obsahujú.

Usúdili sme, že pre účely našej práce nie je použitie tejto metódy vhodné, nakoľko pri malých inštanciách Rush Hour by dekomponovateľné problémy nepredstavovali ťažkú logickú úlohu a boli by vyriešené riešiteľmi ihneď, a naopak pri veľkých inštanciách Rush Hour by nedekomponovateľné problémy boli nevyriešiteľné.

3.2 Analýza stavového grafu

Ďalšou možnou metódou analýzy, je analýza stavového grafu. Vlastnosti stavového grafu sú úzko spojené s vlastnosťami konkrétnej inštancie problému a v tejto kapitole sa zaoberáme skúmaním vlastností problému pomocou analýzy vlastností stavového grafu.

Stavový graf je taký graf, ktorý popisuje všetky možné stavy problému. Teda pre každú možnú platnú konfiguráciu z definície 1.2.3 je v stavovom grafe práve jeden vrchol grafu prislúchajúci tejto konfigurácii.

Ďalej medzi vrcholmi grafu existuje hrana medzi stavmi A a B práve vtedy, keď je možné zo stavu A spraviť ťah do stavu B a naopak, keďže ťahy Rush Hour sú reverzibilné.

Takto zostrojený stavový graf predstavuje akési bludisko, po ktorom sa riešiteľ problému pohybuje. Platné ťahy sú pohyb po hranách grafu a cieľom je dostať sa z vrcholu označeného ako počiatočný do ľubovoľného z vrcholov označených ako cieľových.

Stavový graf zjednodušuje pohľad na problém a jeho analýzu, keďže tvar stavového grafu už zahŕňa všetky pravidlá hry. Avšak dôvod prečo sa takýto graf nevyužíva na riešenie či už počítačom alebo človekom je, že zostrojiť ho je značne výpočtovo náročné a pre väčšie úrovne Rush Hour tento graf je neprimerane veľký.

3.2.1 Vizualizácia

Na jednoduchšie spracovanie dát zo stavových grafov je potrebné ich prehľadne vizualizovať, aby boli prístupnejšie pre ďalšiu analýzu. Vizualizáciou je možné získať ďaleko lepšiu predstavu o povahe dát ako rôznymi inými transformáciami. Hlavným problémom vo vizualizácií je priradiť vrcholom a hranám ich súradnice v ploche alebo priestore, kde premietame graf tak, aby sa zdôraznili vlastnosti grafu. Keďže stavové grafy nami vybraných úrovní boli komplikované a mali priveľa hrán a vrcholov, tak výsledkom nesprávne priradených súradníc by bolo chaotické zobrazenie. Výsledná vizualizácia grafov by bola nečitateľná. Ako prostriedok na vykreslenie grafov sme použili knižnicu MatPlotLib [14] pre Python. Generovanie grafov prebiehalo vo viacerých iteráciách a výsledky sme analyzovali postupne.

Kamada-Kawai kreslenie grafu

Kamada-Kawaii [20] kresliaci algoritmus patrí do triedy grafových kreslení založených na sile. Ich účelom je rozmiestniť uzly grafu v dvoj alebo trojrozmernom priestore tak, aby všetky hrany boli približne rovnaké a čo najmenej sa prekrižovali. Hlavným princípom silových algoritmov sú sily medzi vrcholmi a hranami grafu na princípe pružín. Ďalej sa grafový systém simuluje ako fyzikálny systém telies až dovtedy, kým sa ustáli do nejakého lokálneho minima, ktoré sa považuje za výsledný graf.

Kamada-Kawaii algoritmus natiahne medzi každými dvoma vrcholmi pružinu o veľkosti najkratšej cesty medzi nimi, vo vizualizácií sa snaží grafovo blízke vrcholy držať blízko seba a grafovo ďaleké vrcholy ďaleko od seba.

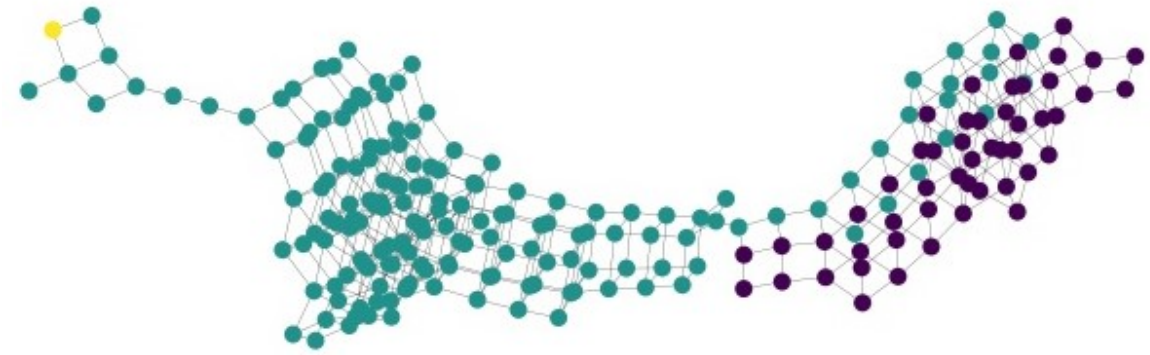
Za hlavné výhody silových algoritmov sa považuje to, že dosahujú veľmi dobré výsledky hlavne pri veľkostiach do 500 vrcholov. Keďže sú založené na princípe analógie k bežným predmetom ako sú pružiny, správanie algoritmov je ľahko pochopiteľné a predvídateľné. Navyše sú flexibilné a ich implementácia nie je náročná. Tiež veľmi dôležitá je ich interaktivita, to znamená, že používateľ môže sledovať ako sa graf vyvíja. Hlavnou nevýhodou týchto algoritmov je ich časová náročnosť.

Grafy úrovní

Vytvorili sme tri rôzne typy zobrazenia stavového grafu úrovne. Základný graf úrovne nemá v sebe žiadne dáta od používateľov. Grafické zobrazenie však napomáha zobraziť štruktúru úrovne. Každý vrchol grafu predstavuje práve jednu validnú a dosiahnuteľnú konfiguráciu z počiatočnej konfigurácie. Aby sme lepšie zobrazili vzťahy medzi koncovými vrcholmi, pri kreslení stavového priestoru sme výherné stavy nepovažovali za koncové, ale z tohto dôvodu môžu byť niektoré stavy v striktnej definícií Rush Hour nedosiahnuteľné. Vo vizualizácii sme sa rozhodli vrcholy odlíšiť farebne podľa ich významu (Obr. 3.1):

- žltý vrchol je počiatočná konfigurácia,
- fialové vrcholy sú výherné konfigurácie,
- zelené vrcholy sú ostatné konfigurácie.

Každá hrana predstavuje možný ťah medzi konfiguráciami, avšak nie sú vykreslené všetky možné ťahy. Skriptom vykresľujeme len jednoduché ťahy, pretože inak by vyobrazených hrán bolo príliš mnoho, graf by bol nečitateľný a neplnil by svoju základnú funkciu.



Obr. 3.1: Graf stavového priestoru

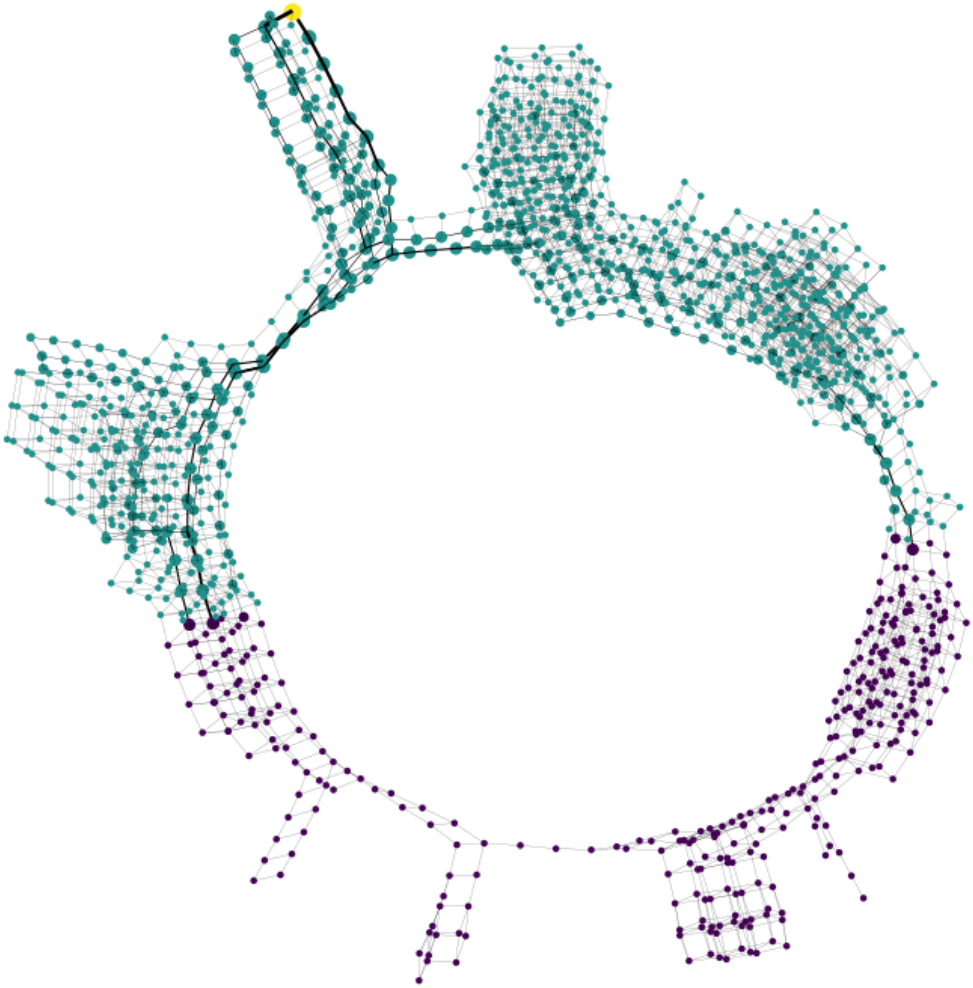
Vážené grafy

Pridaním ďalšieho parametra váhy do grafu vznikajú *vážené grafy*. Grafy je možné vážiť vrcholovo alebo hranovo. V našom prípade sme použili vrcholové aj hranové váženie. Pridali sme dve informácie do grafu o návšteve vrcholov a vznikli dve verzie grafov - *časový* a *návštevový* graf.

Pri návštevovom grafe (Obr. 3.2) je hrúbka každej hrany a veľkosť každého vrcholu prenásobená percentom návštev danej konfigurácie alebo prechodu. Aby sa dali všetky dáta zobraziť, tak sa riešiteľove zložené ťahy rozkladajú na jednotlivé čiastkové jednoduché ťahy. Rozklad je realizovaný jednoducho, zoberie sa najmenší počet jednoduchých ťahov akými je možné realizovať tento zložený ťah.

Pri časovom grafe (Obr. 3.3) sa do úvahy ako váhy neberú návštevy konkrétnych vrcholov alebo hrán grafu ale čas, ktorý riešiteľ strávil v danom vrchole alebo hrane. Navyše pri už spomínanom rozklade na jednoduché ťahy sa rozkladá aj čas zloženého ťahu, rovnomerne na všetky čiastkové jednoduché ťahy.

Tieto dva typy grafov umožňujú prehľad, akým spôsobom sa hráči pohybovali pri riešení problémov. Napríklad, pri ľahších problémoch má väčšina riešení rovnakú cestu riešenia. Takáto vizualizácia nám prináša predstavu o riešení jednotlivých úrovní hráčmi. Keďže vykreslenie grafu nie je triviálna záležitosť a je dôležité, ktorým smerom sú natočené vrcholy, pripravili sme skript na interakciu s grafmi, ktorý je popísaný v



Obr. 3.2: Graf prechodov

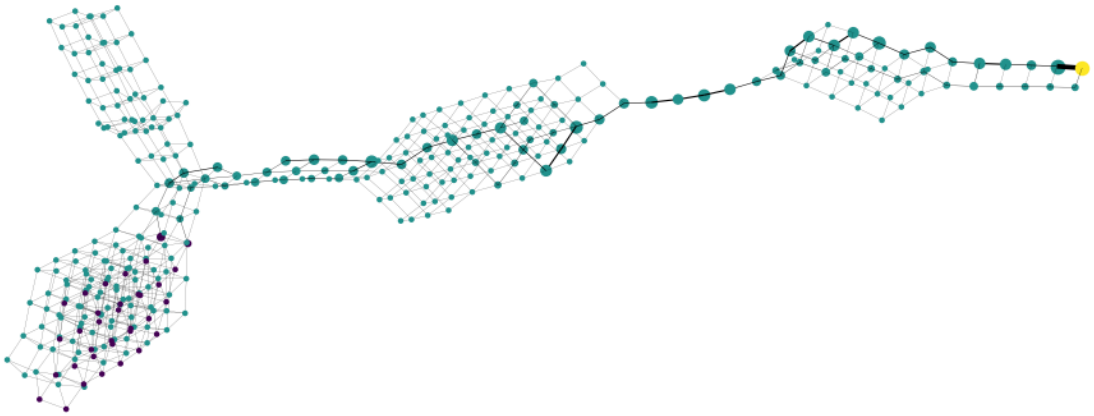
nasledujúcej sekcii.

Generovací skript

Na dynamické vykresľovanie stavových grafov sme implementovali Python skript `network_graphs.py`, ktorý s pomocou knižnice MatPlotLib [14] a mierne upravenej knižnice NetworkX [9] vykresľuje všetky tri typy grafov popísaných v časti Grafy úrovní v tejto kapitole.

Najlepšie rozpoloženia vrcholov grafu bolo výstupom z 3D verzie algoritmu Kamada-Kawaii. Nakoľko vykresľovacia funkcia tejto knižnice nevedela vykresliť 3D graf, bolo nevyhnutné, aby sme NetworkX prispôsobili. Naprogramovali sme `nx_pylab.py`, ktorý je mierne upravenou kópiou súboru z NetworkX tak, aby vykresľoval len 3D grafy.

Generovací skript renderuje grafy všetkých úrovní, zoradených vzostupne podľa



Obr. 3.3: Graf stráveného času

počtu ich konfigurácií. Výstupom je skupina obrázkov stavových priestorov. Keďže tieto obrázky reprezentujú 2D priemet 3D grafov, tak do skriptu sme implementovali funkcionalitu na otáčanie grafov používateľom.

Prvotný pokus bolo využitie experimentálnej podpory vstavanej knižnice Pickle na uloženie a spätné načítanie grafových objektov MatPlotLib, avšak aj keď tento spôsob fungoval, tak načítaný graf stratil nami požadovanú možnosť otáčania. Z uvedeného dôvodu sme museli použiť iné riešenie.

Implementovali sme ukladanie reprezentácie grafu ako `.gml` súboru (grafový modelovací jazyk [11]). Vypočítané pozície vrcholov sú ukladané pomocou knižnice Pickle. Keďže operácie vypočítavajúce pozície vrcholov a hrúbku hrán sú na výpočet značne náročné, skript pred výpočtom skontroluje, či medzivýsledky sú už uložené. V prípade ak áno, tak medzivýsledky načíta a tým sa čas opakovaného generovania grafov skracuje.

Bez dodatkových parametrov príkazového riadku skript len vygeneruje grafy všetkých úrovní a uloží ich obrázky. Je však možné použiť dodatočnú funkcionalitu pomocou nasledovných parametrov príkazového riadku:

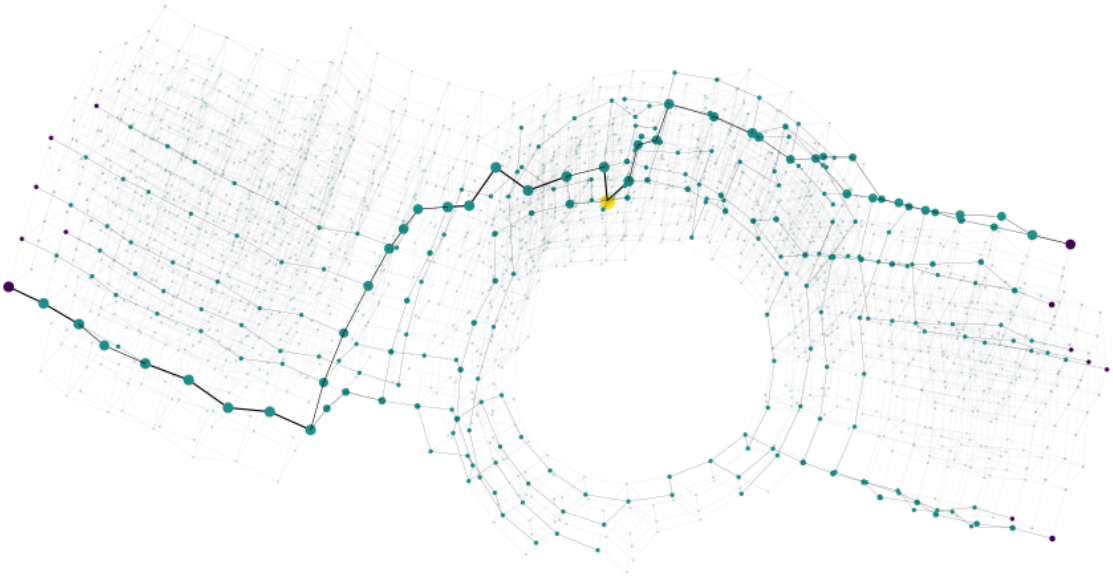
- `-h, -help` - zobrazí pomocníka so zoznamom parametrov,
- `-id <číslo>` - spracuje len úroveň s daným číslom,
- `-v, -view` - pri spracovávaní ľubovoľného grafu otvorí používateľské rozhranie MatPlotLib na prezeranie, približovanie a otáčanie grafu,
- `-bo, -basic-only` - vygeneruje len úrovňové grafy bez váh,
- `-vl, -visited-only` - vygeneruje len vážené grafy podľa návštev,
- `-tl, -timed-only` - vygeneruje len vážené grafy podľa času.

Popisovaný skript spolu s grafmi a vypočítanými údajmi je súčasťou CD, ktoré je priložené k našej práci. Kvôli hardvérovým obmedzeniam, nedostatku operačnej pamäte na výpočet pozícií sú spracované a priložené len grafy 178 úrovní.

3.2.2 Vybrané významné vlastnosti

V tejto sekcií popisujeme niektoré z grafov úrovní, ktoré sme vygenerovali a na základe skúmania našli ich zaujímavé vlastnosti. Z vlastností grafov vieme popísať spôsob riešenia jednotlivých úrovní ľudským riešiteľom. Pre lepšiu viditeľnosť informácií sme na nasledujúcich obrázkoch vykresľovali nenavštívené hrany a vrcholy len veľmi matne. Pomohlo nám to lepšie identifikovať používané hrany, a tým ľahšie zistiť správanie ľudských riešiteľov.

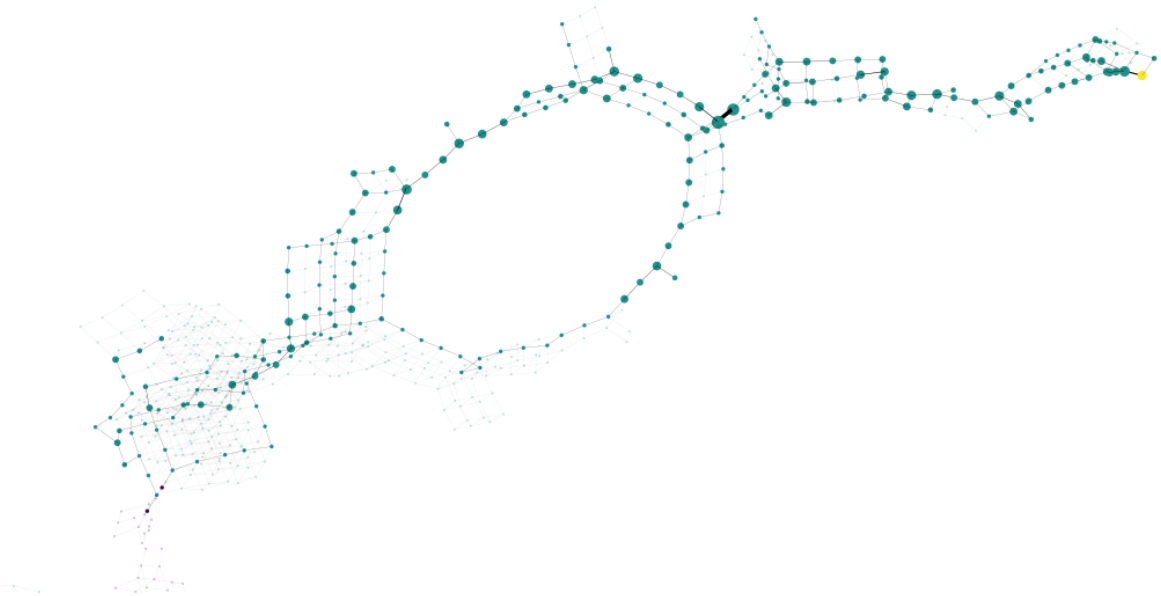
Úroveň 143 Jednou zo zaujímavých úrovní bola úroveň 143. Na *návštevovom* grafe (Obr. 3.4) je možné vidieť, že má dva rôzne zhluky koncových pozícií, a že počiatočná pozícia je približne v strede medzi nimi. Z grafu je zrejmé, že väčšina hráčov riešila logickú úlohu jednou z dvoch rôznych ciest, aj keď existovalo viacero možných riešení. Graf je zaujímavý hlavne z dôvodu, že dve úplne odlišné riešenia sa vyskytujú približne rovnako často a omnoho častejšie ako všetky ostatné riešenia. Táto skutočnosť nasvedčuje tomu, že o riešení sa rozhodlo v prvom ťahu autom a následne takmer všetci hráči riešili úlohu rovnako.



Obr. 3.4: Návštevový graf, úroveň 143

Úroveň 93 Úroveň 93 je ďalšou zaujímavou úrovňou z pohľadu vetvenia. Na *časovom* grafe (Obr. 3.5) je znázornené, že cesta do cieľa sa v istom bode rozvetví na dve odlišné

časti a neskôr sa znovu spojí do jednej cesty ešte pred cieľom. Čo je však zaujímavé, že pred výberom jednej z týchto dvoch ciest sa riešitelia dlho rozhodovali, ktorú cestu zvolia. Poukazuje to na skutočnosť, že riešitelia vedeli identifikovať výrazne odlišné cesty už na ich začiatku a odlíšiť ich od iných, nie tak dôležitých ciest (rozhodnutí) nachádzajúcich sa inde v stavovom priestore. Ďalšie takéto rozhodovanie už v grafe nie je viditeľné, nakoľko v ňom nie je iné rozdelenie na dve výrazne odlišné cesty.

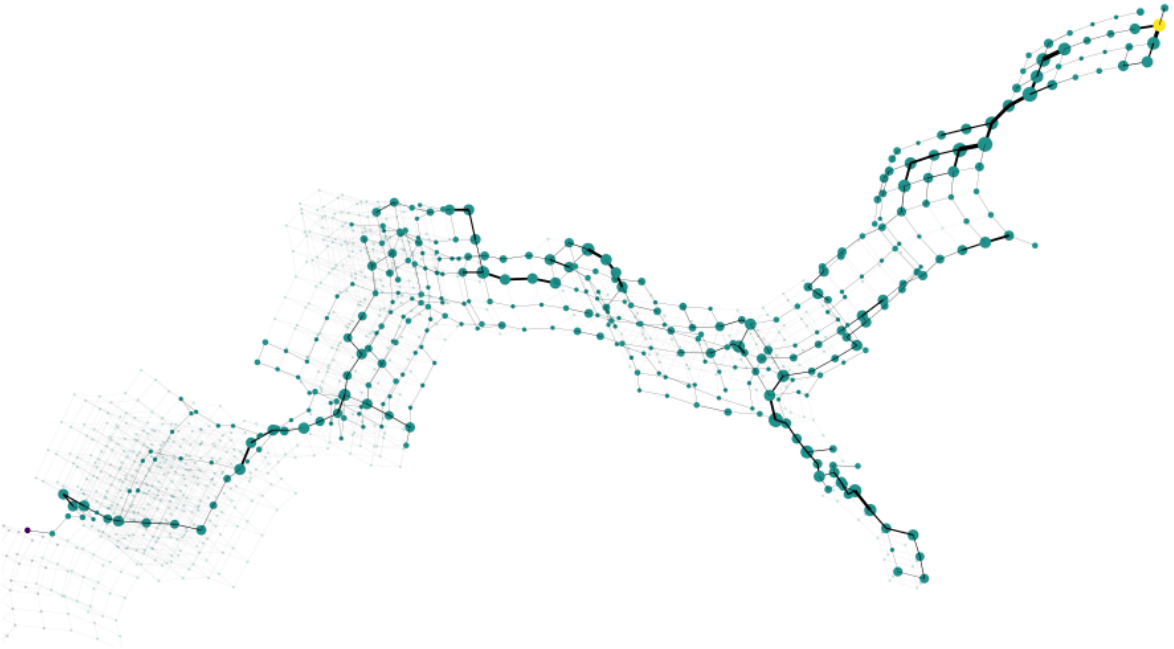


Obr. 3.5: Časový graf, úroveň 93

Úroveň 120 Na *časovom* grafe úrovne 120 (Obr. 3.6) je možné vidieť ďalší zaujímavý prvok stavového priestoru. Približne v polovici cesty od štartu do cieľa je znázornená „slepá ulička“, teda úzky výbežok stavového priestoru, ktorý nikam nevedie. Zo stavového grafu je ľahko viditeľné, že vrcholy slepej uličky nie je potrebné a vhodné navštíviť, napriek tomu riešitelia tieto stavy navštívili a strávili v nich pomerne veľa času.

Na začiatku grafu pri jeho moste je možné si všimnúť časový lievik. Je zrejmé, že tento vrchol vyvolal dlhšie premýšľanie nad ďalším riešením problému. V priebehu analýzy sme sa s podobnými lievikmi stretli pri viacerých úrovniach, čo znamenalo, že pokiaľ z nejakej konfigurácie existovalo veľmi málo ťahov do ďalšej konfigurácie, tak riešitelia dlho premýšľali a hľadali ten správny ťah.

Úroveň 145 Na obrázku 3.7 je zobrazený *návštevový* graf úrovne 145. Túto úroveň sme zaradili medzi ľahšie, je možné ju vyriešiť 12 zloženými ťahmi, počet jej konfigurácií je 2322. Z grafu je viditeľné, že navštívených konfigurácií bolo veľmi málo a to presne 237, čo tvorí sotva 10% celkových konfigurácií, a to aj napriek tomu, že úroveň vyriešilo



Obr. 3.6: Časový graf, úroveň 120

až 50 riešiteľov. Vznikol teda veľký nepomer medzi riešiteľmi preskúmanými a celkovými možnými konfiguráciami.

Najmenší počet jednoduchých ťahov na splnenie úrovne je 22. Spočítali sme, že nastalo minimálne $22 \cdot 50 = 1100$ návštev konfigurácií, z nich aspoň $1 - \frac{237}{1100} = 78.6\%$ bolo duplikátov.

Z uvedeného sme usúdili, že pri ľahších úrovniach veľké množstvo riešiteľov riešilo problém rovnakým spôsobom.

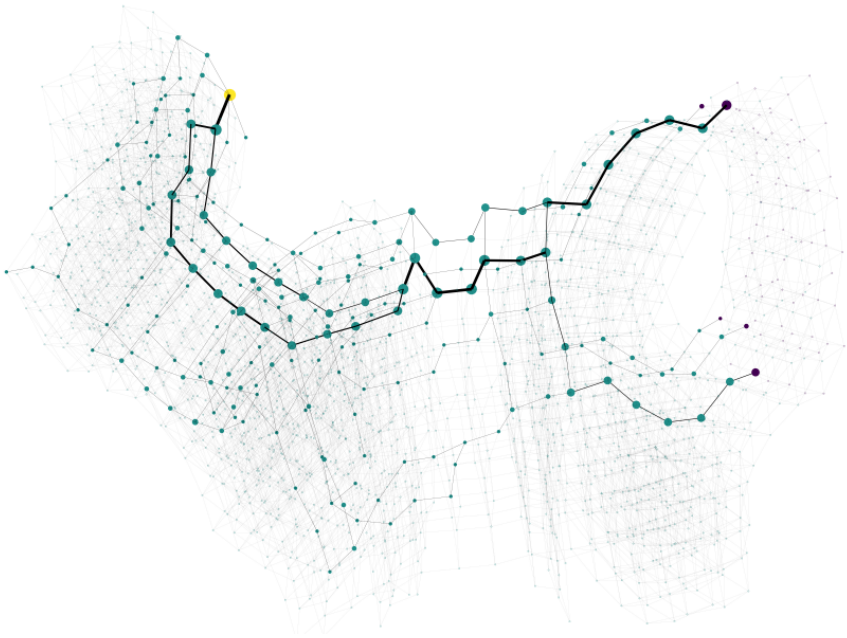
3.2.3 Zhrnutie

Pri skúmaní týchto vlastností na viacerých grafoch iných úrovní sme usúdili, že nie všetky platia všeobecne. Veľké rozvetvenie stavového priestoru sme identifikovali na niekoľkých grafoch, ale iba v polovici z nich bolo možné vidieť hráčov zotrvať na jednom mieste, kde sa rozhodovali, ktorú cestu zvoliť.

Ďalej sme sa zamerali na výrazne rôzne možnosti ťahov na začiatku grafu, ktorými sme sa zaoberali pri úrovni 143. Zistili sme, že zatiaľ čo vo väčšine prípadov takéhoto rozloženia stavového priestoru sa hráči rozhodli riešiť úroveň jednou z dominantných dostupných ciest, jedna cesta riešenia mala tendenciu prevažovať.

Slepá ulička je vlastnosť stavového priestoru, ktorá sa nevyskytuje často. Keď sa v skúmaných úrovniach vyskytla, tak väčšina hráčov ju navštívila a strávila v nej pomerne veľa času.

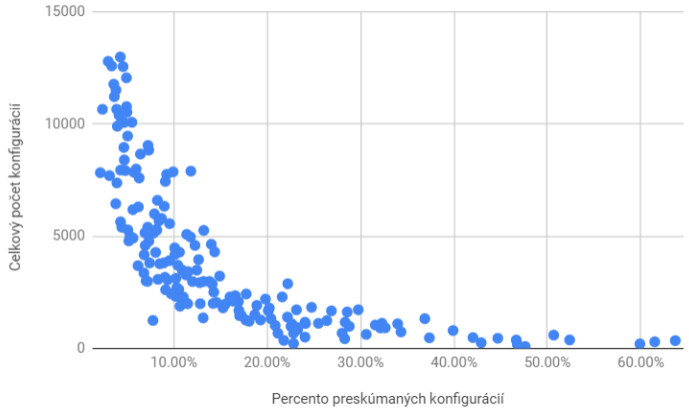
Vizualizácia a následná analýza stavových priestorov nám odhalila, že takmer v po-



Obr. 3.7: Návštevový graf, úroveň 145

lovici skúmaných grafov sa vyskytol časový lievik. Je to zúžená časť stavového priestoru, hráči v stavoch pred ním strávili rozhodovaním viac času. Táto vlastnosť je založená len na tvare stavového priestoru, nie na pravidlách konkrétnej logickej úlohy, čo tvrdia aj Jarušek a Pelánek pri skúmaní hry Sokoban [18].

Ďalej sme skúmali konfigurácie navštívené aspoň jedným riešiteľom v závislosti na iných vlastnostiach úrovne. Náš predpoklad, ktorý sme uviedli pri úrovni 145 sa nám nepotvrdil, ale našli sme zaujímavú závislosť medzi mierou preskúmanej oblasti stavového priestoru a celkovou veľkosťou stavového priestoru. Korelačný graf percenta preskúmanej oblasti oproti počtu celkových konfigurácií je znázornený na obrázku 3.8. Z grafu je zrejmé, že tieto dve vlastnosti sú úzko korelované a percento preskúmanej oblasti stavového priestoru nezávisí na počte hráčov, ktorí danú úroveň riešili.



Obr. 3.8: Korelácia preskúmanej oblasti oproti celkovým konfiguráciám

Kapitola 4

Analýza údajov strojovým učením

Ďalším prostriedok, ktorý sme použili na analýzu zozbieraných údajov bolo strojové učenie. Strojové učenie (Machine learning) je podoblasť umelej inteligencie zaoberajúca sa metódami a algoritmami, ktoré umožňujú programu učiť sa a následne adekvátne reagovať na rôzne vstupné hodnoty iba na základe informácií, ktoré sa naučil bez toho, aby boli reakcie explicitne naprogramované. Algoritmy strojového učenia používajú:

- matematickú štatistiku,
- metódy štatistickej analýzy,
- hĺbkovú analýzu dát (Data mining).

Podľa prístupu sa strojové učenie rozdeľuje na učenie s učiteľom, učenie bez učiteľa a učenie s posilňovaním. V našej práci sme použili učenie s učiteľom. Vo veľkej miere sme nastavovali parametre metód a interpretovali sme výsledky učenia. Pre strojové učenie sme využívali jazyk Python, kde sme používali knižnicu scikit-learn [26]. Knižnica poskytuje veľké množstvo nástrojov na neurónové siete, ako sú implementácie rôznych modelov, operácie s dátami, hodnotiace funkcie a podobne, a je vyvíjaná ako open-source.

Pri analýze vhodnosti použitia strojového učenia na riešenie nášho problému sme usúdili, že vypracované stavové grafy sú príliš komplikované pre väčšinu metód strojového učenia a ich priame napojenie do modelu učenia by nemalo správny účinok. Z uvedeného dôvodu sme vyextrahovali vlastnosti zo stavového priestoru a tieto sú popísané v podkapitole 4.2.

4.1 Model strojového učenia

Aby sme dosiahli čo najlepšie výsledky, pri analýze dát sme použili niekoľko rôznych modelov strojového učenia, ktorých princípy popisujeme v tejto kapitole. Každý z modelov sme natrénovali a následne ohodnotili kvalitu výslednej predikcie. Za účelom

ohodnotenia sme si vstupné dáta rozdelili na trénovacie a testovacie, tak aby sme v čo najväčšej možnej miere zabránili pretrénovanie. Pretrénovanie (overfitting) je jav keď model strojového učenia je príliš dlho trénovaný a postupne dosiahne výbornú úspešnosť na trénovacích dátach, avšak na nových neznámych dátach začne čoraz viac dochádzať ku chybám, stráca sa schopnosť generalizácie.

Hodnotiaca funkcia

Aby sme vedeli hodnotiť ako dobre sa natrénoval daný model strojového učenia, potrebujeme použiť takzvanú hodnotiacu funkciu, ktorá priradí modelu hodnotu vyjadrujúcu ako dobre predikuje vstupné dáta.

My sme použili funkciu $R^2(y, \hat{y})$, ktorá počíta takzvaný koeficient determinácie. Ako parametre berie vektor reálnych výsledkov y a vektor predikovaných výsledkov $\hat{y}$.

Počíta sa nasledovne:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

kde $\bar{y} = \frac{1}{n_{\text{samples}}} \sum_i y_i$

Táto funkcia je dobrým indikátorom presnosti a používa ju aj veľa interných hodnotiacich funkcií scikit-learn.

4.1.1 Metóda podporných vektorov

Metóda podporných vektorov (Support Vector Machine), skratkou SVM predstavuje veľmi dôležitý objav v oblasti strojového učenia. Radí sa medzi najlepšie metódy strojového učenia a to hlavne preto, že je veľmi flexibilná a nedochádza pri nej k závažným chybám na dátach pozbieraných z reálneho sveta. Je široko používaná v mnohých problémoch v reálnom živote, napríklad pri klasifikácii sentimentu v texte a podobne.

SVM je pôvodne klasifikačný algoritmus, ktorý hľadá lineárnu separáciu medzi trénovacími bodmi, z čoho utvorí dve kategórie. Rozhodovanie v ňom prebieha na základe hyperroviny (priamka v 2D, rovina v 3D, atď.). Lineárna separácia je taká hyperrovina, pre ktorú vzdialenosť k najbližším bodom z oboch kategórií je čo najväčšia možná. Inak povedané, je to taká rovina, ktorá zabezpečí najširšiu oddeľujúcu hranicu medzi dvomi kategóriami. Polohu a smer tejto roviny určujú okrajové body v oboch kategóriách. Vzdialenejšie vstupy už nemajú vplyv na smer a polohu roviny. Tieto okrajové body sa volajú *podporné vektory*, podľa ktorých je metóda pomenovaná. Dôležitými parametrami SVM sú *jadro* (kernel) a *C*.

Jadro je funkcia, ktorá je základom SVM. Táto funkcia transformuje vstupné vektory (body) do vyšších dimenzií aby sa zbavila nutnosti lineárnosti SVM. Navyše pre

väčšinu jadier sa nemusia transformované vektory komplikovane zostrojovať a násobiť, čo zaberá príliš veľa času, ale jadrá vedia tieto operácie robiť efektívne. Existuje viac rôznych typov jadier a rozdeľujú sa hlavne podľa toho, či riešia lineárny problém alebo nie. Pre lineárnu SVM existuje jedno jadro a pre nelineárnu SVM viac jadier. Najznámejšie je Radial basis function, skratkou RBF s nekonečným počtom dimenzií.

RBF používa parameter *gamma*, ktorý určuje ako ďaleko od hyperroviny budú mať body vplyv na túto hranicu. V prípade vysokej hodnoty gammy hyperrovinu ovplyvňujú len body, ktoré sú pri nej blízko a vzdialené body majú na hyperrovinu malý vplyv. Pri takejto hodnote sa môže začať hyperrovina zakrivovať a kľukatiť okolo jednotlivých dátových bodov. Ak je hodnota gammy nízka, hyperrovinu ovplyvňujú aj vzdialené body.

Parameter C nastavuje kompromis medzi rovnosťou hyperroviny a počtom správne klasifikovaných trénovacích bodov. Čím väčšie je C, tým viac sa SVM snaží predchádzať nesprávne klasifikovaným bodom, a tak hľadať hyperrovinu, ktorá má menšie chybové rozpätie.

Pri našej analýze sme použili dve SVM jadrá - Radial Basis Function jadro a lineárne jadro. Výsledky a porovnanie natrénovania jadier na našich dátach uvádzame v kapitole 5.2.

4.1.2 Neurónová sieť

Klasickou metódou strojového učenia je Neurónová sieť. Je to výpočtový model založený na vlastnostiach biologických nervových systémov. Neurónová sieť je zložená z viacerých *perceptrónov* (napodobneniny biologických neurónov).

Perceptrón je jednotka, ktorá spracováva viac vstupov a určuje jeden výstup v procese zvanom *aktivácia*. Pre každý vstup má práve jednu váhu, následne pomocou všetkých váh spočíta lineárnu kombináciu vstupov. Na výsledok aplikuje zväčša nelineárnu aktivačnú funkciu, výsledok použije ako výstup. Pri neurónových sieťach sa lineárne aktivačné funkcie väčšinou nepoužívajú, nakoľko výsledkom opakovaného aplikovania lineárnych funkcií je zase len lineárna funkcia. Neurónové siete nastavujú váhy perceptrónov v procese učenia, ktorý môže byť s učiteľom alebo bez učiteľa. Po ukončení tohto procesu sa už jednotlivé váhy nemenia a výpočtový systém produkuje výsledky podľa toho ako sa natrénoval.

Zapojenie perceptrónov navzájom alebo na vstup určuje architektúra siete. Najčastejšie sa používa viacvrstvová architektúra, pri ktorej sa perceptróny členia do postupných vrstiev a dve za sebou idúce vrstvy sa plne prepoja. Výstup zo siete takejto architektúry je výstup poslednej vrstvy.

Neurónové siete sa dnes používajú v rôznych odvetviach. Najčastejšie vyžitie je však pri regulačnej technike, rozpoznávaní obrazov a anomálií v pohyblivom obraze, analýze

dát a v umelej inteligencii. Zložité neurónové siete majú niekoľko rôznych algoritmov na ich natrénovanie. Ako jedny z najlepších trénovacích algoritmov sú napríklad Adam alebo RMS Prop, ktoré sú vylepšeniami známeho Gradient Descent algoritmu, fungujúceho na princípe hľadania parametra, ktorý má najväčší vplyv na minimalizačnú funkciu (chybu v predikcii).

My sme použili dve z najrozšírenejších aktivačných funkcií neurónových sietí, a to hyperbolický tangens a rektifikovaná lineárna jednotka, skratkou ReLU.

Hyperbolický tangens

Hyperbolický tangens je aktivačná funkcia využívaná najmä preto, lebo silno negatívne vstupy mapuje na -1 a silno pozitívne vstupy mapuje na 1, pričom si zachováva nenulovú deriváciu. Tiež má dobrú deriváciu v blízkosti 0.

$$f(x) = \tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$$

Rektifikovaná lineárna jednotka

Rektifikovaná lineárna jednotka je jedna z najnovších aktivačných funkcií. Rozšírila sa najmä s používaním sietí hĺbkového učenia, pretože samotná ReLU a aj jej derivácia je výpočtovo nenáročná, a aj napriek tomu nelineárna. Avšak ReLU má jeden problém. Ak vstup je záporný, tak záporný už zostane počas celého učenia, pretože derivácia ReLU v záporných vstupoch je nulová. Toto sa nazýva *problém umierajúcej ReLU*. Na vyriešenie uvedeného problému sa používa deravá ReLU, ktorá má v záporných číslach lineárnu funkciu s malým sklonom namiesto konštantnej nuly. Tým sa problémová nulová derivácia v zápore zmení na veľmi nízku deriváciu.

$$f(x) = \begin{cases} \alpha x & x < 0 \\ x & x \geq 0 \end{cases}$$

Deravá ReLU je parametrizovaná funkcia, ktorá je lineárna pre $x \geq 0$, ale jej sklon pre $x < 0$ je α-násobne menší, kde α je parameter tejto funkcie. Má zachované všetky potrebné vlastnosti ReLU, ale keďže má nenulovú deriváciu, tak problém umierajúcej ReLU sa nevyskytne.

4.1.3 Hrebeňová regresia

Hrebeňová regresia (Ridge Regression) je regresia zameraná na multikolineárne alebo skoro multikolineárne vzťahy. Multikolinearita znamená, že vstupné premenné sú navzájom od seba lineárne závislé. Pri takýchto vzťahoch bežné metódy (napríklad metóda najmenších štvorcov) príliš dobre nefungujú, keďže aj malá zmena vo vstupných

dátach spôsobí veľkú zmenu vo výslednej funkcii. V takomto prípade sú bežné metódy veľmi nestabilné.

Hrebeňová regresia sa snaží túto nestabilitu riešiť tým, že zámerne zavedie vychýlenie (bias) a takto zníži chybu predikcie. Metóda je pomerne kontroverzná, keďže väčšina ostatných metód sa snaží vychýlenie eliminovať ale my sme ju použili, pretože exceluje v prípadoch, kedy väčšina ostatných modelov má príliš veľkú chybovosť.

4.1.4 Náhodný les

Náhodný les je skupinová predikčná metóda na klasifikáciu a regresiu. Na predikciu používa model rozhodovacieho stromu a pomocou metódy bootstrap agregácie sa snaží zabrániť pretrénovaniu tohto modelu. Model náhodného lesa sme použili ako najznámejšieho reprezentanta skupinových predikčných metód.

Rozhodovací strom

Rozhodovací strom je zložený z viacerých rozhodovacích vrcholov, z ktorých každý reprezentuje jeden *test* na dátach. Každý vrchol má dve podvetvy reprezentujúce výsledok jeho testu.

Príklad testu: vo vrchole je $x < 5$, a ak atribút x je menší ako 5, tak ďalej výpočet prebieha v ľavej vetve, inak bude prebiehať v pravej. Takto sa pokračuje, až pokým sa výpočet nedostane na *dátový* vrchol, ktorý obsahuje numerickú hodnotu predikcie.

Nevýhodou tejto metódy je, že nevie dobre generalizovať, pretože má tendenciu sa pretrénovávať (príliš sa prispôsobiť trénovacím dátam). V snahe čo najviac zabrániť pretrénovaniu rozhodovacieho stromu sa stromom nastavuje maximálna hĺbka listov.

Rozhodovacie stromy sa trénujú pomerne jednoducho, v každom kroku vyberieme test, ktorý maximalizuje zníženie v rozptyle výsledných dvoch skupín dát. Ďalej rekurzívne trénujeme obe vzniknuté vetvy stromu. Ak nám v niektorom vrchole ostáva už iba jeden dátový bod alebo sme dosiahli maximálnu hĺbku, tak namiesto rozhodovacieho vrcholu vložíme dátový, ktorého hodnotu nastavíme na hodnotu dátového bodu, respektíve ich priemeru.

Bootstrap agregácia

Bootstrap agregácia je metóda, ktorú náhodný les používa na redukovanie miery pretrénovávania rozhodovacích stromov. V metóde sa namiesto jedného modelu učenia trénuje n modelov a ich výstupy sa priemerujú. Takto sa ľahko zabráni príliš veľkým výkyvom v predikcii.

Keďže rozhodovací strom je deterministická metóda učenia, tak sa každému z vytvorených modelov dávajú iné trénovacie dáta. Z východiskových trénovacích dát sa

rovnomerne náhodne a s opakovaním vyberajú dátové body, až kým sa nedosiahne požadovaná veľkosť pre čiastkové dáta.

4.2 Extrahované vlastnosti

V tejto časti popisujeme vlastnosti, ktoré sme extrahovali pre každú úroveň hry. Extrahované vlastnosti boli následne postúpené strojovému učeniu za účelom analýzy. Každú z vlastností sme zakódovali do n-dimenzionálneho vektoru. Následne sme vektory spojili do jednej matice, ktorú sme použili ako vstup pre výpočet.

4.2.1 Konfigurácie a minimálne riešenie

Dve tradičné vlastnosti, ktoré sa uvádzajú pri zadaniach Rush Hour sú počet konfigurácií a počet ťahov minimálneho riešenia. Počet konfigurácií sme jednoducho zisťovali zostrojením stavového grafu, keďže táto vlastnosť je práve počet vrcholov grafu. Princíp stavového grafu sme popísali v kapitole 3.2.

Po zostrojení grafu je pomerne triviálne zistiť počet ťahov minimálneho riešenia jednoduchým prehľadávaním do šírky s vypĺňaním minimálnej vzdialenosti od počiatočného vrcholu do každého vrcholu grafu. Potom stačí prejsť všetky koncové vrcholy, a nájsť ten s najmenšou vzdialenosťou.

Zadefinovali sme si čo v hre znamená jednoduchý a čo zložený ťah. Pretože každý z nich má iný počet ťahov minimálneho riešenia, pre minimálne riešenie sme zadefinovali až dve vlastnosti - minimálne jednoduché riešenie a minimálne zložené riešenie.

4.2.2 Mosty stavového grafu

Ďalšia zo skúmaných vlastností grafu bola most stavového grafu - hrana, ktorú keby sme odstránili, tak by už neexistovala cesta medzi počiatočným vrcholom a hociktorým z koncových vrcholov.

Existencia mostu zaručuje, že každý riešiteľ úlohy musí most nájsť na to, aby dosiahol riešenie. Ak by nájdenie mostu z počiatočného stavu bolo príliš jednoduché, tak existencia mostu by vôbec neprispela k obtiažnosti problému. Preto sme do úvahy nebrali mosty, ktoré boli príliš blízko pri počiatočnom vrchole.

Most grafu môžeme zovšeobecniť na problém minimálneho rezu grafu, teda nájdenie takej množiny hrán, ktorú keď odstránime, tak sa graf rozpadne na dve časti. Jedna z verzií problému minimálneho rezu grafu navyše definuje zdrojový a koncový vrchol, pre ktoré po rozpadnutí musí platiť, že zdrojový vrchol sa musí nachádzať v jednej časti a koncový vrchol v druhej časti grafu. Na riešenie uvedenej verzie problému sme si vybrali jeden z efektívnych algoritmov, ktoré ju riešia.

4.2.3 Hustota stavového grafu

Hustota grafu je indikátor popisujúci početnosť hrán grafu v pomere so všetkými možnými hranami, teda koľko percent hrán existuje zo všetkých možných kombinácií vrcholov.

Počíta sa ako

$$d = \frac{2m}{n(n-1)}$$

kde m je počet hrán grafu a n je počet jeho vrcholov.

Hustota môže naznačovať ako ľahko je možné sa pohybovať v stavovom priestore, čím väčšia hustota, tým ľahšie sa môžeme dostať z bodu A do bodu B, nakoľko je tam menej obmedzení. Naopak, pri menšej hustote sa vieme dostať z počiatočného vrcholu ku koncovému menším počtom hrán, a teda hranie pomocou metódy chaotického posúvania áut po hracom pláne je menej efektívne.

4.2.4 2-spojité komponenty

2-spojité komponenty definujeme pomocou 2-spojitých grafov. Spojitý graf je 2-spojitý ak platí, že ak odstránime hocijaký vrchol, tak spojitým ostane. 2-spojitý graf neobsahuje žiadny vrchol, ktorý oddeľuje dve časti grafu. Tento typ grafu bol využiteľný v našej práci hlavne preto, lebo potvrdzuje, že neexistuje žiadna konkrétna konfigurácia, ktorú hráč musí prejsť v každom riešení.

2-spojitých komponent je maximálny 2-spojitý podgraf všeobecného grafu. Z počtu takýchto komponentov v grafe úrovne sme vytvorili vlastnosť a predpokladali sme, že čím viac bude komponentov, tým bude úroveň náročnejšia.

Komponenty sme počítali v lineárnom čase pomocou prehľadávania stromu do hĺbky. Počas prehľadávania sme si pamätali takzvaný minimálny bod pre každý podstrom prehľadávania, čo je minimálna hĺbka susedov všetkých vrcholov podstromu, vrátane jeho koreňa. Následne sme hľadali taký vrchol, ktorý má syna s minimálnym bodom väčším alebo rovným ako hĺbka vrcholu. Takže vieme, že vrchol, ktorý sme našli ani žiadny vrchol z jeho pod-stromu nemá suseda zo zvyšnej časti stromu, preto vrchol rozdeľuje graf na viac 2-súvislých komponentov. Nemôže sa tak stať, že sused vrcholu by bol z iného podstromu rovnakej hĺbky, lebo by to znamenalo, že sme ho už museli navštíviť v tomto podstrome prehľadávania do šírky.

Počiatočný bod prehľadávania rozdeľuje graf na komponenty práve vtedy, ak má dvoch alebo viac synov, keďže to znamená, že žiadny z jeho podstromov nemá medzi sebou spojenie.

4.2.5 Komunitné rozdelenie

V štúdií rôznych sietí nie len grafových sa často vyskytuje takzvaná komunitná štruktúra, kedy sa vrcholy zoskupujú do jednotlivých komunít. Pre komunity platí, že vrcholy sú viac pospájané medzi sebou v komunite ako s vrcholmi mimo komunity. Práve pre túto vlastnosť komunít sme sa rozhodli ich použiť na zostavenie ďalších vlastností učenia.

Na hľadanie komunitného rozdelenia sme použili Clauset-Newman-Mooreov pažravý algoritmus [6], pretože je rýchly aj na väčších grafoch. Niektoré úrovne majú v stavovom grafe aj 50 tisíc vrcholov. Algoritmus definuje *modularitu* ako hodnotu podielu hrán vedúcich v rámci jednej komunity s hranami, ktoré vedú medzi komunitami. Od tohto podielu sa ešte odpočíta hodnota modularity, ktorá by nastala v úplne náhodnom grafe.

Algoritmus začína s každým vrcholom vo svojej vlastnej komunite a potom pažravo hľadá dve komunity, ktorých spojením by sa najviac zvýšila modularita. Tento postup opakuje dovtedy, pokiaľ už nevie viac zvýšiť modularitu. Zostávajúce komunity nazve komunitným rozdelením grafu.

Ťahov medzi nájdenými komunitami bude výrazne menej ako ťahov vnútri komunít, teda je pravdepodobnejšie že riešiteľ zostane v komunite ako to, že z nej odíde. Zadefinovali sme dve rôzne vlastnosti - počet komunít v rozdelení stavového grafu a dĺžka cesty od štartu po koniec meraná v komunitách, ktoré musí hráč prejsť.

Avšak komunitné rozdelenie závisí na hranách v grafe, teda rozdelenie bude iné, keď bude počítané na stavových grafoch iba s jednoduchými ťahmi alebo na grafoch so zloženými ťahmi. Rozhodli sme sa každú z týchto vlastností spočítať na oboch typoch grafov, a tak sme vytvorili štyri rôzne vlastnosti pre učenie.

4.2.6 Čas riešenia problému

Ďalej sme vytvorili vlastnosť, ktorá je predikovaná modelmi. Vlastnosť musela odrážať obtiažnosť problému pre ľudských riešiteľov, preto sme zvolili prirodzenú metriku obtiažnosti - čas potrebný na riešenie ľuďmi.

Nakoľko čas riešenia rovnakých úrovní rôznymi riešiteľmi je rozdielny, zozbierané dáta o čase sme museli agregovať. Najskôr sme ako agregáciu pre každú úroveň spočítali medián časov riešení. Na takto pripravených dátach sa však modely neboli schopné natrénovať.

Ako najlepšia agregácia sa ukázal priemer logaritmov času, dôvody popisujeme v kapitole 5.1. Na grafoch priložených v prílohe 1 sme zobrazili vzťah tejto agregácie s ostatnými extrahovanými vlastnosťami. Z grafov sme usúdili, že niektoré vlastnosti majú silnú koreláciu s agregáciou a modely učenia budú schopné sa na agregácií natrénovať. Na grafe minimálnych potrebných zložených ťahov je možné si všimnúť, že

dáta sú mierne skreslené výberom úrovní. Do portálu sme vybrali viac úrovní s väčším počtom minimálnych potrebných ťahov.

4.3 Nastavovanie modelov

Pri návrhu modelov strojového učenia je potrebné myslieť na to, že prvotné nastavenia závisia hlavne od druhu siete a jej zamerania. Pri niektorých môže ísť o počet vrstiev, pri iných o počet buniek skrytých vrstiev. Pri výbere počtov vrstiev a neurónov sme zvažovali aj následný dopad na funkciu siete. Pri narastajúcom počte prepojení medzi bunkami sa mení schopnosť siete zovšeobecňovať. Počet takýchto prepojení je závislý na počte dát, ktoré sú určené k učeniu siete.

V tejto podkapitole sme sa zaoberali nastavením hyperparametrov s ohľadom na rýchlosť a presnosť učenia. Parametre sme kontrolovali pomocou krížovej validácie, čo nám umožnilo trénovať model na väčšej množine dát. Hodnoty hyperparametrov sme optimalizovali pomocou metódy prehľadávania mriežky. S použitím modelov z najlepšími parametrami sme následne vyberali najdôležitejšie vlastnosti.

4.3.1 Hyperparametre

Modely strojového učenia často obsahujú množstvo takzvaných hyperparametrov, ktoré ovplyvňujú ako sa bude natrénovaný model správať. Hyperparametrami nazývame také parametre modelu strojového učenia, ktoré sa nevedia sami naučiť a je potrebné ich ručne nastaviť. Dôvodom môže byť, že parameter ovplyvňuje spôsob učenia alebo nepredvídateľne mení už naučené informácie. Napríklad, v modely neurónovej siete je hyperparameter rýchlosť učenia, architektúra neurónovej siete (počet vrstiev, počet neurónov na vrstvách), chybová funkcia a podobne.

Nastavenie jedného parametra nepredstavuje väčšinou zložitý problém, ale pri nastavovaní viacerých parametrov je potrebné myslieť na to, že sa môžu navzájom ovplyvňovať, a tým môže vzniknúť viacrozmerný priestor. Z tohto dôvodu sa používajú optimalizačné algoritmy, ktoré prechádzajú priestor hyperparametrov s cieľom nájsť optimálne nastavenie.

4.3.2 Metóda krížovej validácie

Pri nastavovaní hyperparametrov vzniká problém, že model je možné pretrénovať. Prílišným prehľadávaním hyperparametrov by sme spôsobili, že na konkrétnych trénovacích dátach pomocou hyperparametrov model pretrénujeme. Štandardne by bolo potrebné trénovacie dáta rozdeliť na trénovacie a *validačné*. Následne by sa na validačných dátach hodnotilo, ako vhodne sme hyperparametre zvolili, a tým predišli pretrénovaniu

modelu.

Týmto spôsobom by sme však odstránili značnú časť dát z pôvodného datasetu. Navyše spôsob rozdelenia dát na testovací a validačný set by mohol značne ovplyvniť akým spôsobom sa model natrénuje, keďže časti pôvodného datasetu by boli použité na validáciu, nie na trénovanie.

Riešením uvedením problémov je *krížová validácia*. Validácia je proces, kedy sa trénovacie dáta rozdelia na k rovnako veľkých častí. Vyberieme jednu z častí ako validačnú, model natrénujeme na zvyšných $k-1$ častiach a na validačnej časti vypočítame skóre modelu. Následne vyberieme inú časť ako validačnú a model natrénujeme znovu. Proces opakujeme, až pokým nepoužijeme všetky časti ako validačné. Výsledné skóre pre model sa potom vyjadrí ako priemer čiastkových validačných skóre.

Výhoda tohto postupu je, že sa neplytvá dátami na validačný set, nevýhoda je, že ide o výpočtovo náročný proces, nakoľko model je potrebné natrénovať k-krát.

4.3.3 Metóda prehľadávania mriežky

Prehľadávanie mriežky (Grid Search) je metóda, ktorá sa zaoberá optimalizovaním hyperparametrov. Princíp spočíva v úplnom prehľadávaní všetkých možných kombinácií *prípustných hodnôt* hyperparametrov optimalizovaného modelu. Samotnému prehľadávaniu mriežky musíme ako hyperparameter nastaviť prípustné hodnoty hyperparamtrov podradeného modelu, aby sme obmedzili výpočtovú zložitosť optimalizovania.

Existujú rôzne variácie mriežkového prehľadávania ako napríklad rekurzívne, pri ktorom sa po nájdení najlepších parametrov pokračuje s ďalším *jemnejším* prehľadávaním v okolí najlepších výsledkov predchádzajúcej iterácie.

V našej práci sme použili krížovo validované prehľadávanie mriežky, ktoré na výpočet skóre daných parametrov používa metódu krížovej validácie.

4.3.4 Metóda rekurzívneho eliminovania vlastností

Metóda rekurzívneho eliminovania vlastností (Recursive feature elimination) skratkou RFE je proces na získavanie *dôležitých* vlastností modelu. Je možné ho aplikovať na modely strojového učenia, ktoré poznajú dôležitosť svojich vstupných vlastností. Napríklad pre lineárnu SVM sa táto dôležitosť spočíta z koeficientov natrénovanej hyperroviny.

Táto metóda najskôr natrénuje model na všetkých vlastnostiach, nájde najmenej dôležitú vlastnosť podľa hodnotenia modelu a odstráni ju. Na takto upravených vlastnostiach sa rekurzívne pokračuje, až pokým sa počet vlastností nezredukuje na požadované množstvo.

V prípade, že je potrebné zistiť optimálny počet parametrov, je možné použiť okrem RFE aj krížovú validáciu. Takto upravená metóda sa nazýva Rekurzívne eliminovanie

vlastností s krížovou validáciou, skratkou RFECV, ktorá postupne pomocou RFE znižuje počet parametrov a kontroluje skóre krížovou validáciou. Proces sa opakuje, až dovtedy kým nenastane výrazný pokles v skóre krížovej validácie, čo indikuje, že počet parametrov je už príliš malý.

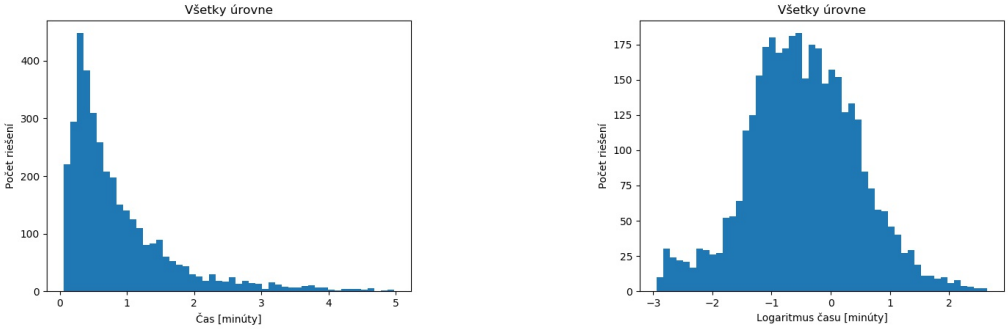
Kapitola 5

Výsledky

V tejto kapitole popisujeme akým spôsobom boli rozložené dáta a ako sme ich upravili pre potreby učenia. Na upravené dáta sme aplikovali metódy strojového učenia uvedené v kapitole 4. Overili sme kvalitu predikcie a následne sme identifikovali najvýznamnejšie vlastnosti.

5.1 Distribúcia dát

Keďže väčšina metód strojového učenia predpokladá, že distribúcia všetkých vlastností sa podobá na normálnu distribúciu, znormalizovali sme vstupné vlastnosti tak, aby mali priemer 0 a štandardnú odchýlku 1. Pri analyzovaní spôsobu ako je potrebné upraviť výstupnú vlastnosť čas, sme zostrojili histogram distribúcie času všetkých riešení (Obr. 5.1a). Distribúcia času mala priebeh podobný logaritmickej krivke, preto sme zostrojili histogram logaritmu času (Obr. 5.1b), ktorý má už normálne rozdelenie. Následne sme zostrojili histogramy časov riešení a logaritmov časov pre jednotlivé úrovne. Vybrané histogramy úrovní sú priložené v prílohe 2. Z týchto histogramov a analýz zaoberajúcich sa podobnou problematikou [8] sme usúdili, že čas riešení logickej úlohy Rush Hour má log-normálnu distribúciu. Preto po zlogaritmovaní času riešenia sme vyrobili premennú, ktorá mala normálne rozdelenie a bola použiteľná na strojové učenie.



(a) Histogram času (b) Histogram logaritmu času

Obr. 5.1: Histogramy času riešenia všetkých úrovní

5.2 Prehľadávanie mriežky

Pomocou metódy prehľadávania mriežky opísaného v kapitole 4.3.3 sme trénovali niekoľko hyperparametrov našich modelov. Prehľadávanie mriežky používalo krížovú validáciu s hodnotou $k = 5$. Možné hyperparametre sme vyberali väčšinou v logaritmickej škále a to nasledovne:

- SVR s jadrom RBF,
  - 50 hodnôt `C` na logaritmickej škále od 10^{-3} do 10^{3},
  - 50 hodnôt `gamma` na logaritmickej škále od 10^{-3} do 10^{3},
- SVR s lineárnym jadrom: 50 hodnôt `C` na logaritmickej škále od 10^{-3} do 10^{3},
- Hrebeňová regresia: 50 hodnôt `alpha` na logaritmickej škále od 10^{-3} do 10^{3},
- Náhodný les: hodnoty `n_estimators` zo zoznamu $[1, 3, 5, 7, 9, 11, 13, 15]$,
- Neurónová sieť ReLU: 1 alebo 2 skryté vrstvy a na každej 10 hodnôt na lineárnej škále od 10 do 100 neurónov,
- Neurónová sieť Tanh: 1 alebo 2 skryté vrstvy a na každej 10 hodnôt na lineárnej škále od 10 do 100 neurónov.

Na skórovanie modelu prehľadávanie mriežky využívalo hodnotiacu funkciu R^2 popísanú v kapitole 4.1. Najlepšie hyperparametre pre každý model, ich trénovacie a testovacie skóre R^2 sú zobrazené v tabuľke 5.1.

Model	Najlepšie parametre	Trénovacie skóre	Testovacie skóre
SVR jadro RBF	C: 0.869, gamma: 0.010	0.650	0.633
SVR jadro linear	C: 0.013	0.628	**0.648**
Hrebeňová regresia	alfa: 44.984	0.647	0.645
Náhodný les	n_estimators: 13	**0.929**	0.614
Neurónová sieť ReLU	skryté neuróny: 2x 10	0.787	0.382
Neurónová sieť Tanh	skryté neuróny: 1x 10	0.678	0.639

Tabuľka 5.1: Skóre učenia metódou prehľadávania mriežky

Na trénovacích dátach dosiahol najlepší výsledok model náhodného lesa, ale mal nízke skóre na testovacích dátach. Tento druh modelu má tendenciu sa pretrénovať, teda príliš sa prispôsobiť konkrétnym dátam, a ani krížová validácia tomu nevie úplne zabrániť, ako sa potvrdilo aj na našich výsledkoch.

Najlepšie výsledky celkovo dosahuje model SVR s lineárnym jadrom a ako druhé najlepšie hrebeňová regresia. Keďže obidva tieto modely podporujú hodnotenie vlastností, mohli sme ich použiť na vyhľadanie najvýznamnejších vlastností. Modely schopné analýzy vlastností sa natrénovali rovnako dobre, dokonca lepšie ako ostatné modely a dostatočne vystihujú nami nájdené korelácie. Pri ďalšom skúmaní sme sa preto zamerali už iba na modely SVR s lineárnym jadrom, hrebeňovú regresiu a náhodný les.

5.3 Rekurzívne eliminovanie vlastností

Na identifikáciu najdôležitejších vlastností problému sme použili metódu rekurzívneho eliminovania vlastností popísanú v kapitole 4.3.4. Pomocou metódy sme spracovali modely s najlepšími hyperparametrami, ktoré sme určili pomocou prehľadávania mriežky. Trénovacie a testovacie skóre modelov na redukovaných vlastnostiach sú uvedené v tabuľke 5.2.

Model	Trénovacie skóre	Testovacie skóre
SVR jadro linear	0.626	**0.640**
Hrebeňová regresia	0.641	0.633
Náhodný les	**0.908**	0.595

Tabuľka 5.2: Skóre učenia RFECV

Porovnaním hodnôt z tabuľky 5.1 a tabuľky 5.2 je zrejmé, že redukcia vlastností pomocou metódy RFECV výrazne neznížila skóre modelov na trénovacej a ani na

testovacej množine dát. Rozdiely v skóre sú prijateľné a vyradené vlastnosti nemali významný dopad na predikciu.

Model	SVR jadro linear	Hrebeňová regresia	Náhodný les
Min. počet zložených ťahov	**0.224**	**0.233**	**0.477**
Min. počet jednoduchých ťahov	**0.245**	**0.326**	**0.393**
Počet konfigurácií	n/a	n/a	0.081
Veľkosť minimálneho rezu	-0.035	n/a	n/a
Hustota hrán grafu	n/a	n/a	n/a
Počet 2-spojitých komponentov	n/a	n/a	0.049
Počet jednoduchých komunít	n/a	n/a	n/a
Vzdialenosť jednoduchých komunít	n/a	n/a	n/a
Počet zložených komunít	0.052	n/a	n/a
Vzdialenosť zložených komunít	0.031	n/a	n/a

Tabuľka 5.3: Natrénované vlastnosti RFECV

V tabuľke 5.3 sú uvedené interné koeficienty modelov pre každú vlastnosť. Ak v tabuľke je hodnota n/a, tak vlastnosť bola vyradená z modelu ako málo významná.

Z tabuľky je zrejmé, že dve najdôležitejšie vlastnosti sú minimálny počet ťahov a minimálny počet jednoduchých ťahov, ktoré najviac ovplyvňujú modely. Sekundárny dopad na predikciu majú počet zložených komunít a vzdialenosť zložených komunít, najviac ovplyvňujú SVR s lineárnym jadrom.

Model	SVR jadro linear	Hrebeňová regresia	Náhodný les
Min. počet zložených ťahov	**1**	**1**	**1**
Min. počet jednoduchých ťahov	**1**	**1**	**1**
Počet konfigurácií	6	8	**1**
Veľkosť minimálneho rezu	**1**	4	5
Hustota hrán grafu	3	5	3
Počet 2-spojitých komponentov	5	7	**1**
Počet jednoduchých komunít	*2*	*2*	4
Vzdialenosť jednoduchých komunít	4	9	6
Počet zložených komunít	**1**	6	*2*
Vzdialenosť zložených komunít	**1**	3	7

Tabuľka 5.4: Dôležitosť vlastností RFECV

V tabuľke 5.4 je uvedené hodnotenie vlastností podľa RFECV. Najvýznamnejšie vlastnosti majú v tabuľke hodnotu 1, klesajúca významnosť vlastností je v tabuľke

znázornená narastajúcou hodnotou. Táto hodnota znamená reverzné poradie, v akom bola vlastnosť eliminovaná. Pre účely lepšej čitateľnosti sme zvýraznili hodnoty 1 a 2.

5.4 Zhrnutie

Našim výskumom sme zistili, že obtiažnosť Rush Hour závisí na tvare stavového priestoru. Zo stavového priestoru predikujeme obtiažnosť pomocou strojového učenia na takmer 65%. Ako najlepší model na predikciu sme vyhodnotili SVR s lineárnym jadrom.

Identifikovali sme, ktoré vlastnosti stavového priestoru sa najviac podieľajú na obtiažnosti problému. Ako najvýznamnejšie sú primárne vlastnosti - minimálny počet jednoduchých ťahov a minimálny počet zložených ťahov. Sekundárne vlastnosti, ktoré najviac ovplyvňujú obtiažnosť problému sú počet jednoduchých komunít a počet zložených komunít. Zaujímavé je, že zatiaľ čo počty komunít sú významné vlastnosti vo všetkých modeloch, tak počet konfigurácií stavového grafu je málo významný, napriek tomu, že počet komunít úzko závisí na počte konfigurácií.

Záver

V našej práci sme sa zaoberali otázkou ako ľudia riešia ťažké logické úlohy so zameraním na obtiažnosť problému pre ľudského riešiteľa v súvislosti s atribútmi zadania problému. Ako konkrétne inštancie problémov pre naše skúmanie sme si zvolili úrovne logickej hry Rush Hour, ktorá je dostatočne výpočtovo ťažká. V práci sme popísali formálnu definíciu problému Rush Hour a rozobrali jeho zložitosť s dôkazom, že sa jedná o PSPACE-úplný problém. Pri štúdiu danej oblasti a možných postupov riešenia sme zistili, že problematika bola a je predmetom skúmania viacerých výskumov. Väčšina z nich však popisovala problematiku na logickej hre Sokoban.

Na účely skúmania ako ľudia riešia logické úlohy je potrebná množina údajov, ktorá musí spĺňať určité kvantitatívne ale aj kvalitatívne parametre. Vzorku údajov s dostatočnými atribútmi sa nám nepodarilo získať z externých zdrojov, preto sme na zbieranie údajov vytvorili portál s hrou Rush Hour. Každý ťah riešiteľa pri riešení problému sme zaznamenávali a týmto spôsobom sme sa snažili zozbierať čo najväčšie množstvo dát. Portál je stále v prevádzke a dáta pre ďalšie možné analýzy sa neustále rozširujú.

Zozbierané dáta sme podrobili skúmaniu, počas ktorého sme použili viacero analytických metód. Metóda dekompozície nebola pre riešenie nášho problému vhodná, nakoľko nebolo možné zostaviť takú sadu úloh, kde by sa efekt dekomponovateľnosti riešiteľmi dal skúmať. Pre pochopenie riešenia sme jednotlivé úrovne vizualizovali do stavového priestoru. Čiastkovým výstupom tejto práce je aj program na vykresľovanie a manipuláciu 3D grafov stavového priestoru. Grafy môžu byť navyše ováhované časovou alebo návštevovou zložkou. Vo vykreslených grafoch sme zistili niektoré zaujímavé vlastnosti a tie sme sa snažili zovšeobecniť, čo sa nám pri niektorých podarilo.

Jedným z cieľov práce bolo zaoberať sa otázkou, či a ako dobre vieme predpovedať čas potrebný na riešenie inštancie zo syntaktických vlastností danej inštancie. Na riešenie problematiky sme použili strojové učenie. Pre predpoveď obtiažnosti úrovne (času riešenia) však nami zozbierané dáta neboli postačujúce, preto sme ich doplnili o dáta z portálu UmímeMatiku. Za účelom analýzy syntaktických vlastností tvaru stavového priestoru sme vybrali niekoľko modelov strojového učenia. Použitím rôznych metód učenia sme modely natrénovali a identifikovali sme významné vlastnosti. Vyhodnotili sme, že najlepší model na trénovacích dátach bol náhodný les, ale na testovacích dátach nedosahoval najlepšie výsledky. Ako najvhodnejší model sa ukázal Support Vector

Machine s lineárnym jadrom, ktorý dosahoval takmer 65% úspešnosť na testovacích dátach. V závere môžeme skonštatovať, že sa nám podarilo splniť tento cieľ práce, nakoľko obtiažnosť problému Rush Hour vieme predikovať s presnosťou takmer 65%.

Identifikovať významné vlastnosti bolo ďalším cieľom našej práce. Potvrdilo sa nám, že primárne vlastnosti, ktoré najviac ovplyvňujú zložitosť riešenia problému ľudským riešiteľom sú minimálny počet jednoduchých ťahov a minimálny počet zložených ťahov, čo je zrejmé už aj z prvého pohľadu na problém zložitosti. Dôležitým prínosom našej práce však bolo, že sme identifikovali významné sekundárne vlastnosti - počet jednoduchých komunít a počet zložených komunít stavového priestoru, čím môžeme potvrdiť, že aj tento cieľ práce sa nám podarilo naplniť.

Výsledky našej práce je možné rozšíriť bližším skúmaním vlastností počtu komunít, keďže tieto vykazujú značnú mieru korelácie s obtiažnosťou problému a ich mierne modifikácie by pravdepodobne výrazne zvýšili mieru predikcie. Miera predikcie by sa tiež dala zvýšiť skúmaním ďalších vlastností stavového priestoru, vzhľadom na to, že nami skúmané vlastnosti nevysvetľujú celú varianciu obtiažnosti problému.

Ďalším rozšírením výskumu danej problematiky by mohlo byť skúmanie väčších inštancií problému, čo by prinieslo rozšírené možnosti stavového priestoru a v stavových grafoch by sa mohli vyskytnúť nové významné vlastnosti. Výsledky výskumu by boli použiteľné najmä v oblasti vzdelávania a psychológie za účelom analyzovania spôsobu logického myslenia človeka alebo v zábavnom priemysle pre automatické zostavovanie očakávane obtiažnych inštancií problému.

Literatúra

[1] Rush Hour: What to do when you can't find someone to play against!
URL <https://www.thegameaisle.com/rush-hour/> (získané 27.4.2019)

[2] Django Software Foundation: Django (verzia 2.2) [počítačový software]. 2019.
URL <http://djangoproject.com>

[3] Ashlock, D. A.; Schonfeld, J.: Evolution for automatic assessment of the difficulty of sokoban boards. In *IEEE Congress on Evolutionary Computation*, IEEE, 2010, s. 1–8.
URL <http://dblp.uni-trier.de/db/conf/cec/cec2010.html#AshlockS10>

[4] Babinčák, P.: *Analýza chování lidí při řešení Nurikabe*. Dizertačná práca, Masarykova univerzita, Fakulta informatiky, 2011.

[5] Bockholt, M.; Zweig, K. A.: Why Is This So Hard? Insights from the State Space of a Simple Board Game. *Serious Games Lecture Notes in Computer Science*, 2015: str. 147–157, doi:10.1007/978-3-319-19126-3_13.

[6] Clauset, A.; Newman, M. E.; Moore, C.: Finding community structure in very large networks. *Physical review E*, ročník 70, č. 6, 2004: str. 066111.

[7] Flake, G. W.; Baum, E. B.: Rush Hour is PSPACE-complete, or "Why you should generously tip parking lot attendants". *Theoretical Computer Science*, ročník 270, č. 1-2, 2002: s. 895–911.

[8] Forišek, M.: *Theoretical and Practical Aspects of Programming Contest Ratings*. Dizertačná práca, Comenius University Bratislava, 2009.

[9] Hagberg, A. A.; Schult, D. A.; Swart, P. J.: Exploring Network Structure, Dynamics, and Function using NetworkX. In *Proceedings of the 7th Python in Science Conference*, editace G. Varoquaux; T. Vaught; J. Millman, Pasadena, CA USA, 2008, s. 11 – 15.

[10] Hearn, R. A.; Demaine, E. D.: PSPACE-Completeness of Sliding-Block Puzzles and Other Problems through the Nondeterministic Constraint Logic Model of

Computation. *CoRR*, ročník cs.CC/0205005, 2002.
URL <http://arxiv.org/abs/cs.CC/0205005>

[11] Himsolt, M.: GML: A portable graph file format. *Universität Passau*, 1997.
URL <https://www.fim.uni-passau.de/fileadmin/files/lehrstuhl/brandenburg/projekte/gml/gml-technical-report.pdf>

[12] Hipp, D. R.; Kennedy, D.; Mistachkin, J.: SQLite (verzia 3.21.0) [počítačový software].
URL <https://www.sqlite.org/download.html>

[13] Hoffmann, L.: *Puzzles Old and New*. Hamley's Magical Saloons, 1893.

[14] Hunter, J. D.: Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, ročník 9, č. 3, 2007: s. 90–95, doi:10.1109/MCSE.2007.55.

[15] Jarušek, P.; Pelánek, R.: What Determines Difficulty of Transport Puzzles? 2011.
URL <https://www.aaai.org/ocs/index.php/FLAIRS/FLAIRS11/paper/view/2518>

[16] Jarusek, P.; Pelánek, R.: Umíme matiku.
URL <https://www.umimematiku.cz/> (získané 24.4.2019)

[17] Jarusek, P.; Pelánek, R.: Difficulty Rating of Sokoban Puzzle. *Frontiers in Artificial Intelligence and Applications*, ročník 222, 01 2010: s. 140–150, doi: 10.3233/978-1-60750-675-1-140.

[18] Jarusek, P.; Pelánek, R.: Human Problem Solving: Sokoban Case Study. 04 2010.
URL <https://www.fi.muni.cz/adaptivelearning/documents/sokoban.pdf>

[19] Kallaspriit: HTML canvas [počítačový software]. 2013.
URL <http://html-canvas-lib.sourceforge.net/> (získané 24.4.2019)

[20] Kamada, T.; Kawai, S.; aj.: An algorithm for drawing general undirected graphs. *Information processing letters*, ročník 31, č. 1, 1989: s. 7–15.

[21] Klembarová, B.: *Complexity of solving puzzles*. Master thesis, Comenius University, Faculty of Mathematics, Physics and Informatics, Bratislava, 2018.

[22] van Kreveld, M. J.; Löffler, M.; Mutser, P.: Automated puzzle difficulty estimation. *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, 2015: s. 415–422.

[23] Lehni, J.; Puckey, J.: Paper.js (verzia 0.11.8) [počítačový software]. 2011.
URL <http://paperjs.org> (získané 24.4.2019)

[24] Levine, B.; Craik, F. I.: *Mind and the frontal lobes: Cognition, behavior, and brain imaging*. OUP USA, 2012, 274 s.

[25] Michael, F.: Rush Hour.
URL <https://www.michaelfogleman.com/rush/>

[26] Pedregosa, F.; Varoquaux, G.; Gramfort, A.; aj.: Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, ročník 12, 2011: s. 2825–2830.

[27] van Rijn, J.: Rush Hour is PSPACE-complete. 2011.

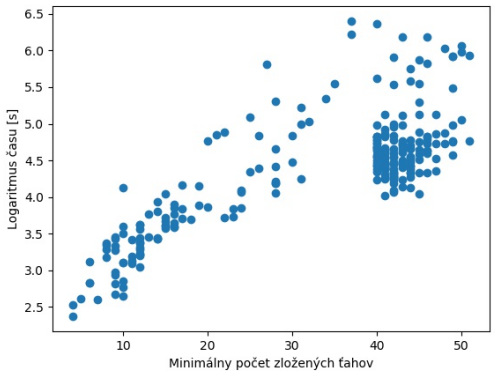
[28] Salesforce: *Heroku*, 2007.
URL <https://www.heroku.com/> (získané 24.4.2019)

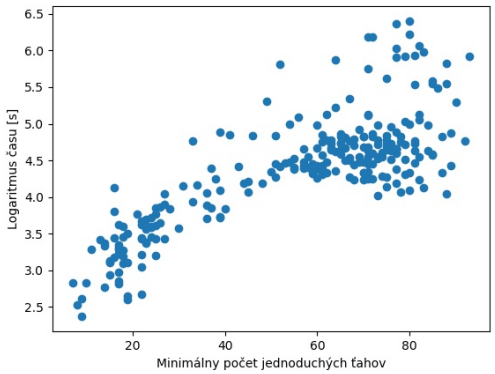
[29] Savitch, W. J.: Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, ročník 4, č. 2, 1970: s. 177–192.

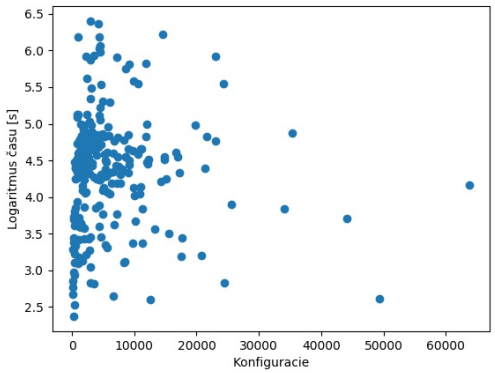
[30] Thomas, G.; Jones, G.; HO, C.: PythonAnywhere.
URL <https://pythonanywhere.com> (získané 24.4.2019)

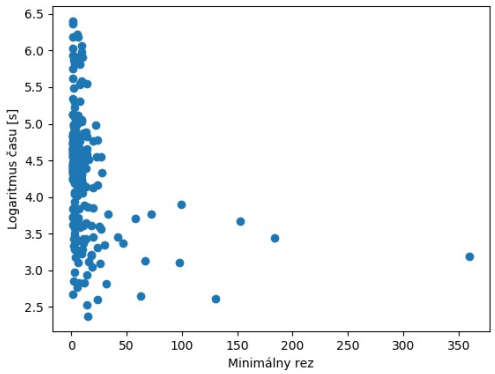
[31] Zaytsev, J.; Kienzle, S.; Bogazzi, A.: Fabric.js (verzia 2.4.6) [počítačový software].
URL <http://fabricjs.com/> (získané 24.4.2019)

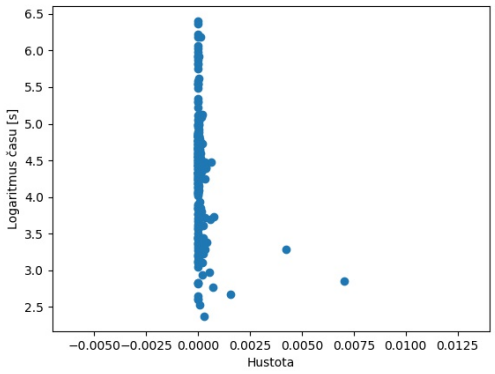
Príloha 1
Grafy závislosti času na extrahovaných vlastnostiach

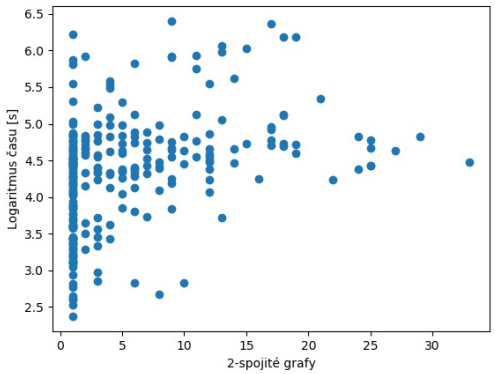


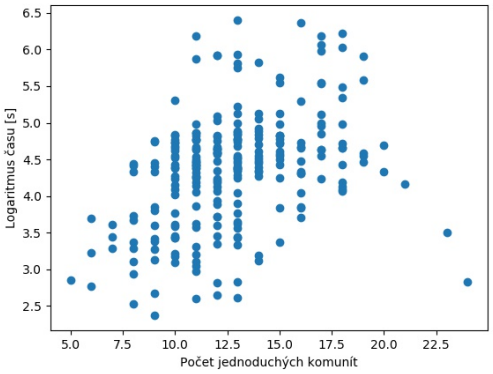


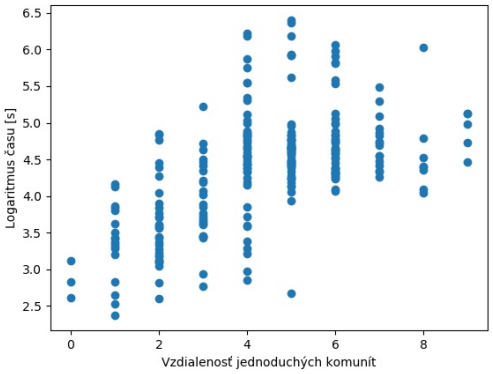


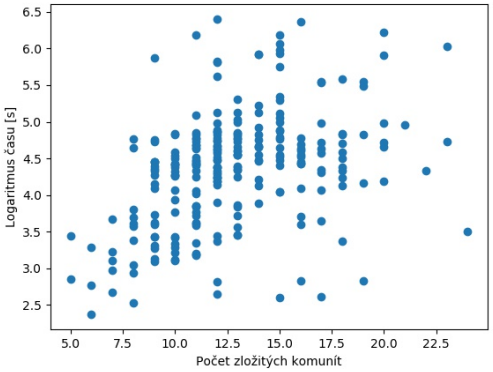


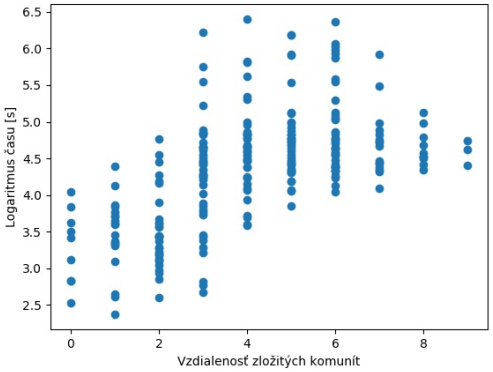




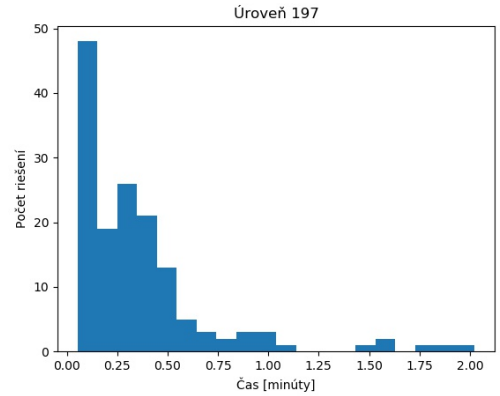

Logaritmus času [s]
Počet jednoduchých komunít

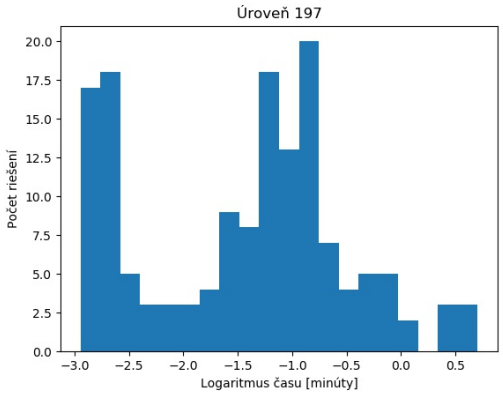

Logaritmus času [s]
Vzdialenosť jednoduchých komunít

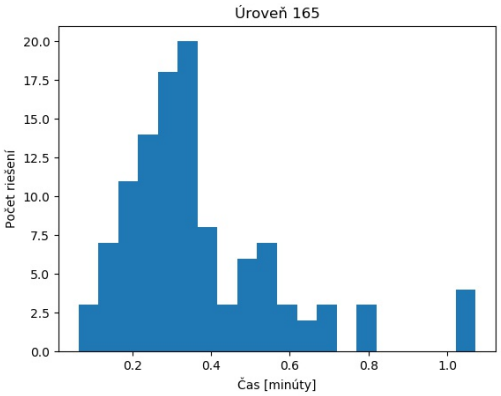

Logaritmus času [s]
Počet zložitých komunít

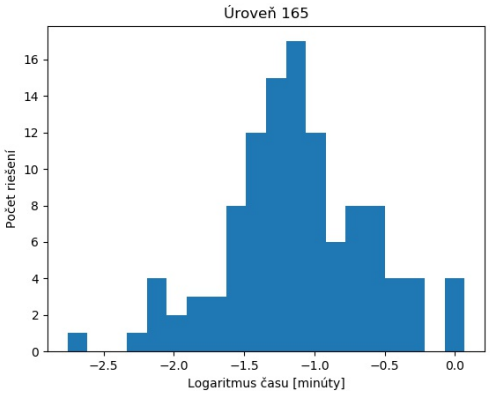

Logaritmus času [s]
Vzdialenosť zložitých komunít

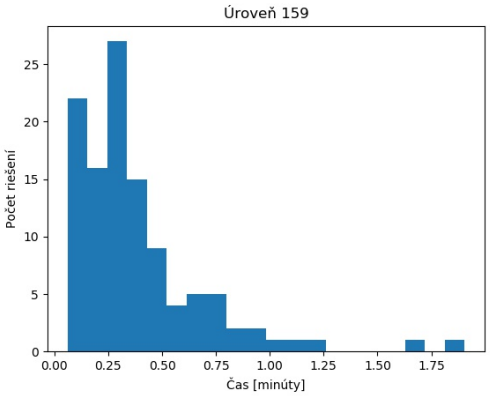
Príloha 2
Histogramy času riešenia a logaritmu času riešenia

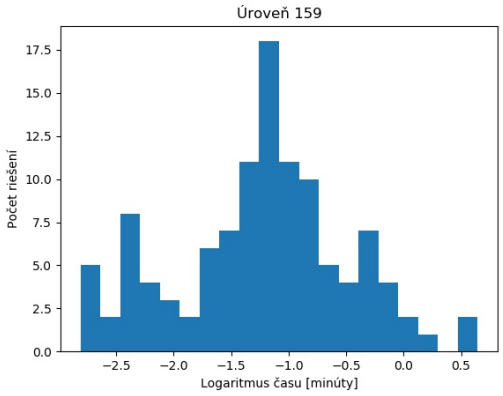


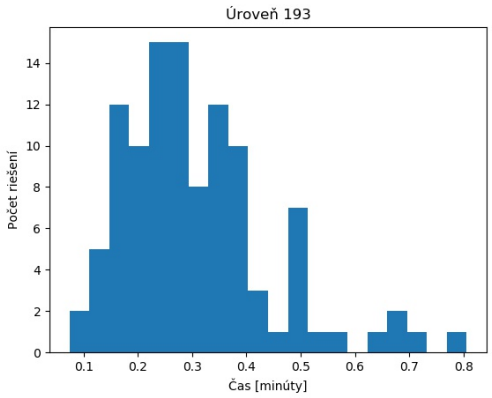


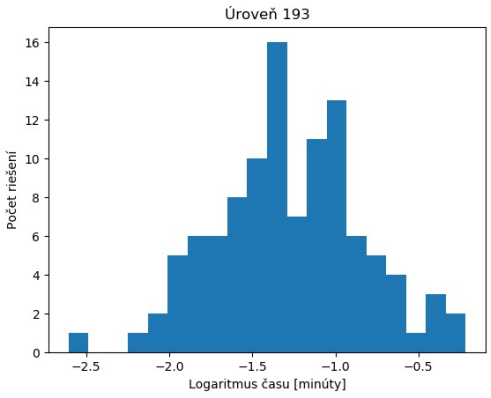


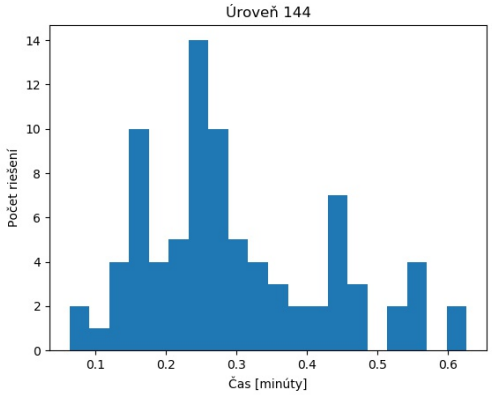


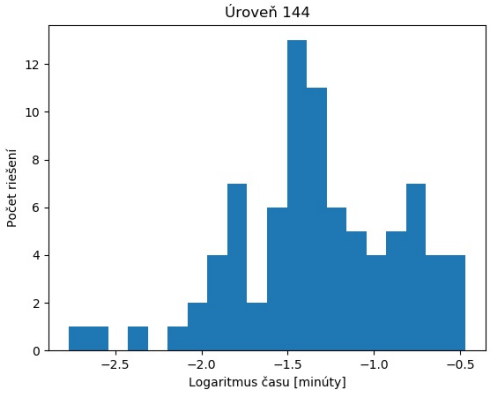


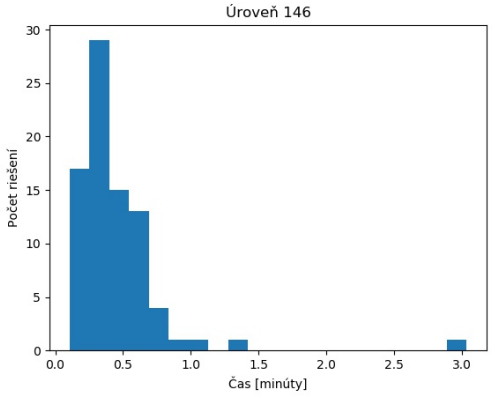


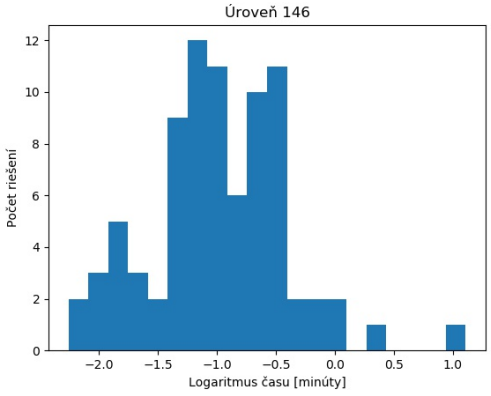

Úroveň 193
Počet riešení
Čas [minúty]


Úroveň 193
Počet riešení
Logaritmus času [minúty]


Úroveň 144
Počet riešení
Čas [minúty]


Úroveň 144
Počet riešení
Logaritmus času [minúty]


Úroveň 146
Počet riešení
Čas [minúty]


Úroveň 146
Počet riešení
Logaritmus času [minúty]

Príloha 3
Zdrojový kód a dáta

Zdrojový kód práce je na priloženom CD. Obsahuje program vykreslujúci 3D grafy, program vyberajúci najlepšie vlastnoti a webový portál na zber dát. CD taktiež obsahuje inštrukcie k programom a dva datasety použité v práci.