FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ DIPLOMOVÉ PRÁCE

Název:	Aplikace pro demonstraci funkce Tabu prohledávání
Student:	Bc. Jaroslav Veselý
Vedoucí:	doc. Ing. Petr Fišer, Ph.D.
Studijní program:	Informatika
Studijní obor:	Webové a softwarové inženýrství
Katedra:	Katedra softwarového inženýrství
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Vytvořte aplikaci pro demonstraci funkce Tabu prohledávání. Aplikace bude použita pro výuku předmětu MI-PAA.
Základní pedagogické požadavky jsou:
- Přehledné a snadno použitelné grafické rozhraní, aplikace musí být jednoduše spustitelná, bez nutnosti jakékoliv instalace.
- Možnost výběru z několika problémů k řešení, automatické i ruční generování instancí problémů, možnost práce s více instancemi, vracení se k nim.
- Možnost nastavování parametrů Tabu prohledávání, možnost nastavování velikosti krátkodobé a dlouhodobé paměti, apod.
- Sledování běhu algoritmu (graficky), vytváření statistik.

Nejprve proveďte důkladnou analýzu funkčních i ostatních (nefunkčních) požadavků. Za tímto účelem proveďte uživatelský průzkum mezi absolventy předmětu MI-PAA a jeho cvičícími.

Dále navrhněte vhodnou SW architekturu pro implementaci. Zde mějte na zřeteli univerzálnost a rozšiřitelnost (pro další problémy).
Nakonec nástroj adekvátně otestujte, proveďte důkladné uživatelské testy.

Seznam odborné literatury

Dodá vedoucí práce.

Ing. Michal Valenta, Ph.D.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 22. ledna 2018

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra softwarového inženýrství

Diplomová práce

Aplikace pro demonstraci funkce Tabu prohledávání

Bc. Jaroslav Veselý

Vedoucí práce: doc. Ing. Petr Fišer, Ph.D.

7. května 2018

Poděkování

Děkuji vedoucímu práce doc. Ing. Petru Fišerovi, Ph.D. za velmi přínosné konzultace a věcné připomínky, především po odborné stránce. Dále děkuji Michalu Kluzáčkovi a Adamu Kuglerovi za spolupráci, díky které vznikla celková aplikace složená z jednotlivých implementačních částí. Také děkuji všem testerům za jejich čas a ochotu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 7. května 2018 .....

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2018 Jaroslav Veselý. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí a nad rámec oprávnění uvedených v Prohlášení na předchozí straně, je nezbytný souhlas autora.

Odkaz na tuto práci

Veselý, Jaroslav. *Aplikace pro demonstraci funkce Tabu prohledávání.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Práce se zabývá návrhem, implementací a uživatelským testováním aplikace pro demonstraci algoritmu Tabu prohledávání. Aplikace má sloužit studentům jako studijní nástroj pro pochopení logiky algoritmu. Uživatelé mají k dispozici na výběr několik problémů, jejiž instance budou mít možnost řešit. Můžou si také nastavovat parametry algoritmu. Aplikace obsahuje vizualizaci průběhu algoritmu a možnost porovnání více běhů. Součástí aplikace je také generátor instancí.

Klíčová slova Tabu prohledávání, webová aplikace, UI, JavaScript, Vue

Abstract

This thesis deals with design, implementation and user testing of application for demonstration of Tabu Search algorithm. Primary goal is to create a tool for students to help them understand how the algorithm works. Several problems are available and users can solve its instances. It is also possible to adjust algrithm parameters. Application offers progress visualisation and comparison of past runs. Instance generator is also available in the application.

Keywords Tabu search, web aplication, UI, JavaScript, Vue

Obsah

Seznam obrázků

Seznam tabulek

Seznam zrojových kódů

Úvod

Cílem práce je vytvoření aplikace, která zjednoduší výuku pokročilých iterativních algoritmů, konkrétně Tabu prohledávání. Hlavním účelem je demonstrace průběhu algoritmu tak, aby ho studenti magisterského programu mohli efektivně pochopit.

Výsledná aplikace bude obsahovat také algoritmus simulovaného ochlazování [1] a genetický algoritmus [2]. Každému z těchto algoritmů se věnuje samostatná diplomová práce.

Aplikace má společný základ, především uživatelské rozhraní a nezávislou implementaci problému a algoritmu. Má být jednoduše rozšiřitelná o další problém a případně algoritmus. Součástí bude tedy návrh komunikace pomocí jednotného rozhraní, které oddělí logiku problému od logiky algoritmu.

Každá práce se zaměřuje na konkrétní problémy související především s daným algoritmem. Každý problém bude samostatný logický celek a při spojení do jedné aplikace by se měly navzájem doplňovat. To platí především pro společnou část uživatelského rozhraní.

Navenek bude vše vypadat jako jedna aplikace, která má tři sekce, nicméně každá sekce odpovídá jednomu pokročilému iterativnímu algoritmu.

Kapitola 1

Tabu prohledávání

Tabu prohledávání je algoritmus, který lze řadit mezi pokročilé iterativní metody.

Pojem „tabu search“ vznikl ve spojitosti s „meta-heuristikou“ v rámci práce „Future paths for integer programming and links to artificial intelligence“ publikované v roce 1986, jejímž autorem je Fred Glover [3]. Prezentoval v ní aplikaci algoritmu na různé známé problémy, kde tento algoritmus v několika případech nacházel řešení velmi blízko optimálnímu a řadil se mezi nejefektivnější [4].

1.1 Myšlenka algoritmu

Tabu prohledávání je specifické svým postupem, který počítá s lokálními optimy a nebere je jako bariéry bránící nalezení lepšího řešení [5]. Další specifická vlastnost je práce s pamětí. U algoritmů jako simulované ochlazování nebo genetické algoritmy se nehledí na historii, ale z aktuálního stavu se pomocí daného pseudonáhodného postupu generuje příští stav. U Tabu se postupuje stavovým prostorem více systematicky pomocí dynamicky se měnícího přípustného okolí stavu. Tabu vychází z předpokladu, že chytrý algoritmus využívá přizpůsobivou paměť, aby mohl dynamicky reagovat na aktuální situaci za pomoci „znalostí“, získaných v průběhu algoritmu [6]. Díky použití takové paměti pak mohou být implementovány postupy, které stavový prostor procházení ekonomicky a efektivně.

1.1.1 Rysy, kterými může Tabu prohledávání disponovat

Jde především o **přizpůsobivou paměť** a **dynamicky řízené prohledávání**. Důležitý je výsledek práce s kombinací možností u těchto dvou hlavních myšlenek algoritmu. Kompozice rysů samozřejmě závisí na potřebách konkrétní aplikace.

Jednotlivé rysy mají ještě velikou možnost parametrizace, která hraje další klíčovou roli pro vlastnosti dané implementace algoritmu.

Možnost práce s přizpůsobivou pamětí [7]

- Selektivita (včetně strategického zapomínání)
- Abstrakce a dekompozice (díky paměti pro explicitní změny i rysy/atributy změn)
- Časování
  - nedávnost událostí
  - frekvence událostí
  - rozlišování krátkodobé a dlouhodobé paměti
- Kvalita a dopad
  - relativní atraktivita alternativních možností
  - velikost změn ve struktuře
  - velikost změn v důsledku aplikace omezujících podmínek
- Kontext
  - regionální vnitřní souvislosti
  - strukturální vnitřní souvislosti
  - sekvenční vnitřní souvislosti

Možnost práce s dynamicky řízeným prohledáváním [7]

- Strategicky navržené omezení a povolení (tabu podmínky a aspirační kritéria)
- Zaměření na dobré oblasti a znaky přispívající k dobrým řešením (intenzifikace)
- Charakterizace a prozkoumávání nových, slibně vypadajících, oblastí (diverzifikace)
- Nemonotónní prohledávací cesty/vzorce (strategická oscilace)
- Integrace a rozšiřování řešení (path relinking)

Tabu prohledávání lze díky takovémuto množství rysů a vlastností přizpůsobit pro řešení konkrétního problému nebo přímo instancí problému se specifickou vlastností. I přes všechny tyto vlastnosti, které lze přizpůsobovat a nastavovat, může být implementace i velmi jednoduchá, rychlá a poměrně efektivní.

1.2 Aplikace na problém

Tabu prohledávání lze použít v podstatě na libovolný optimalizační problém. Takový problém lze definovat jako optimalizaci $f(x)$ pro všechna $x \in X$. Optimalizace zde znamená minimalizace či maximalizace. Funkce f může být lineární, nelineární nebo i stochastická. X je množina všech možných nastavení konfiguračních proměnných a x je vektor představující konkrétní nastavení takových proměnných.

1.3 Od myšlenky blíže k implementaci

Algoritmus je tedy založen na postupném, převážně systematickém, prohledávání stavového prostoru. Již navštívené stavy jsou po nějakou danou dobu (počet iterací) označeny jako tabu. Stavy označené jako tabu se nemohou stát kandidáty na příští navštívený stav (nesmíme se do nich vrátit, jsou pro nás tabu).

Tabu nemusí být celé stavy, bývá to však základ algoritmu, který je společný pro většinu jeho implementací. Tabu může být jakýkoliv princip/rys změny v souvislosti mezi stavy, který bude po použití zakázáno použít znovu na daný počet iterací.

Někdy je však výhodné povolit stav, který je tabu. Algoritmus se může v průběhu dostat do situace, kdy při aplikování tabu změny dostaneme nejlepší dosud nalezený stav. Podmínky připuštění jinak tabu tahů se nazývají *aspirační kritéria*.

Asi nejdůležitější věcí pro smysluplné prohledávání je tabu paměť, která zamezí zacyklení algoritmu. Může to být například historie stavů, které se nastaví přiměřená velikost, aby se algoritmus nezastavil hned u prvního lokálního extrému.

Dále můžeme definovat, že pokud jsme v konfiguraci změnili i-tou pozici, nebudeme ji například dalších 5 iterací měnit (zpět). Tím omezíme frekvenci změn jednotlivých parametrů konfigurace. Zde je otázka, jestli to je pro daný problém dobře nebo špatně a tomu by mělo odpovídat nastavení jak moc frekvenci omezíme a jestli vůbec.

Paměť lze také využít pro shromažďování statistik, například ovlivnění cenové funkce parametrem konfigurace s danou hodnotou (průměr, minimum, maximum, suma).

Tabu prohledávání není založeno na náhodě, což je poměrně neobvyklé pro řešení složitých problémů (např. ze třídy NP-úplné). Oproti algoritmům jako *simulované ochlazování* a *genetický algoritmus* se tak jedná o revoluční změnu. U Tabu se procházení stavovým prostorem řídí nastavenými pravidly. Pravidla jsou sice dynamická, ale deterministická (i za předpokladu, že by PRNG byl nedeterministický). Algoritmus lze doplnit o prvky založené na PRNG (pseudo random number generator), ale podstata algoritmu se o ně neopírá.

Algoritmus definuje pouze princip postupování stavovým prostorem a nabízí spoustu volnosti v implementaci. Volnost se velmi projevuje při implementaci pro konkrétní problém. Při zaměření na jediný problém je k dispozici velký prostor pro optimalizaci. A nejen pomocí nastavení parametrů, ale také vhodným použitím rysů a vlastností, které algoritmus nabízí.

Na druhou stranu obecná implementace pro více algoritmů, které mají různá specifika není tak přímočará. Lze použít obecné principy, ale například strukturální souvislost je u každého problému jiná a parametrizace není řešením. Na toto téma je volně navázáno v následující sekci.

1.4 Algoritmus pro tuto práci

Pro účely této práce se omezím pouze na některé principy, na kterých lze myšlenku poměrně jednoduše demonstrovat. Dalším důležitým aspektem pro výběr implementace je obsah předmětu MI-PAA, pro který je primárně aplikace určena. Musím také brát ohled na to, že implementace bude řešit více různých problémů, takže se algoritmus nesmí konkretizovat například na počet splněných klauzulí u problému SAT, jelikož by neměl abstraktní analogickou strategii pro problém obchodního cestujícího apod.

Implementace Tabu prohledávání pro tuto práci je ve zjednodušené verzi na ukázce kódu 1. Využívá dva typy paměti a jedno aspirační kritérium. Mimo samotnou logiku algoritmu jako takového jsem ještě optimalizoval operaci zjištění, zda je stav tabu. Tato optimalizace bohužel není v ukázce kódu z důvodu nedostatku místa.

První paměť ukládá navštívené stavy. Počet uložených stavů je nastaven vstupním parametrem. Jako datová struktura je použita fronta, jelikož přesně odpovídá použití. Aktuální stav se zařadí na konec fronty a pokud je překročena velikost, odebere se stav na začátku fronty. Tato paměť může být poměrně veliká a prohledávání fronty pro zjištění, zda obsahuje daný stav, je častá operace, která má lineární složitost. Proto jsem se rozhodl stavy ukládat do JS objektu (který jsem použil a reálně funguje jako HashMap/asociativní pole). Díky tomu má kontrola, zda je stav tabu, konstantní složitost. Pointa je v tom, že fronta se edituje ve vnějším cyklu algoritmu a kontrola, zda je stav tabu, probíhá ve vnitřním cyklu.

Druhá paměť ukládá změnu provedenou v každém tahu po dobu, která je zadána vstupním parametrem. Opět se jedná o frontu, která bude ze svého principu spíše menší velikosti. S touto pamětí souvisí také aspirační kritérium. Pokud je změna tabu, ale provedení změny vede k dosud nejlepšímu nalezenému stavu, pak je tah připuštěn.

Další vstupní parametr je počet sousedních stavů, ze kterého se vybere stav následující. Zadává se v procentech (1 až 100) z počtu možných sousedních stavů.

Nakonec lze zvolit, zda je počáteční stav výchozí nebo náhodný. Výchozí znamená, že je zaručeně stejný pro danou instanci problému. V podstatě se jedná o deterministicky vygenerovaný co nejjednodušeji sestavitelný stav pro daný typ problému a instanci.

Obecně je algoritmus optimalizován na volání výpočtu cenové funkce. Místo volání výpočtu ceny stavu při každém porovnávání se cena stavu, se kterým algoritmu aktuálně pracuje ukládá do pomocné proměnné. Díky této jednoduché optimalizaci se algoritmus zrychlil téměř o polovinu (měřen čas výpočtu).

Dlouhou dobu jsem přemýšlel nad implementací sofistikovanějšího principu, který lze logicky aplikovat obecně na více problémů. Nicméně jsem došel k závěru, že takový princip vymyslet neumím. To však nemusí mít negativní dopad na aplikaci. Tím, že bude Tabu prohledávání velmi přímočaré může způsobit zvýšení zájmu o tento algoritmus. Ze cvičení předmětu MI-PAA (Problémy a algoritmy) si pamatuji, že právě Tabu prohledávání je pro semestrální práce nejméně populární. Toto tvrzení podporuje uživatelské testování, kde jsme se studentů ptali jaký algoritmus implementovali. Doufám, že tato jednoduchost zaujme více studentů, než jen samotný název. Mě osobně zaujal fakt, že jsem si pod pojmem Tabu prohledávání neuměl představit, v čem spočívá logika algoritmu. Takových studentů je dle mého názoru více, ale studijního materiálu, který by byl lehce uchopitelný, je nedostatek. Převažují vědecké práce a je poměrně málo zdrojů demonstrujících jakousi lehkost a volnost implementace.

```
function tabuSearch(problem, iterLimit, neighborsToCheckRatio,
    tabuSize, tabuSizeShort, randomStart) {
  let state = problem.getConfiguration(randomStart);
  let sBest = state;
  let tabuStates = [];        // Queue
  let tabuChanges = [];       // Queue

  for (let n = 0; n < iterLimit; n++) {
    let bestCandidate = getNullState();
    let tabuBestCandidate = getNullState();
    let neighborhood = generateNeighborhood(state.getSize(),
        neighborsToCheckRatio);

    for (let i = 0; i < neighborhood.length; i++) {
      let sCandidate = getNeighbor(state, neighborhood[i]);

      if (tabuChanges.indexOf(i) !== -1 && sCandidate.cost < sBest.
          cost) continue;

      if (sCandidate.cost >= bestCandidate.cost) {
        if (!containsSearch(tabuStates, sCandidate)) {
          bestCandidate = sCandidate;
        } else if (compareTabuIndexes(tabuStates, sCandidate,
            tabuBestCandidate) > 0) {
          tabuBestCandidate = sCandidate;
        }
      }
    }

    if (isNullState(bestCandidate)) {
      if (!isNullState(tabuBestCandidate)) {
        state = tabuBestCandidate;
      }
    } else {
      state = bestCandidate;
    }

    if (state.cost > sBest.cost) {
      sBest = state;
    }

    updateTabuStates(tabuStates, tabuSize, state);
    updateTabuChanges(tabuChanges, tabuSizeShort, bestCandidate);
  }

  return sBest;
}
```

Zdrojový kód 1: Zjednodušená verze použité implementace Tabu prohledávání

Kapitola 2

Analýza požadavků a architektura

2.1 Sběr požadavků na aplikaci

Pro sestavení požadavků na aplikaci jsme vytvořili formulář zaměřený na studenty a především absolventy předmětu MI-PAA. Zdrojem společných otázek byly především konzultace s vedoucím a se cvičícími. Jelikož jsem předmět také absolvoval, mám představu, co bych od aplikace očekával. Je ale potřeba podložit vlastní domněnky a ověřit si, co by budoucím studentům usnadnilo pochopení fungování algoritmů a s čím by se jim dobře pracovalo.

2.1.1 Formulář pro sběr požadavků od studentů

Formulář má nejprve společnou část, kde jsou otázky zaměřené na platformu aplikace jako celku a kontext používání aplikace. Vzhledem k tomu, že pro simulované ochlazování a genetický algoritmus již existovaly Java applety, ptali jsme se i na jejich používání. Snažíme se získat důvody a případy použití, abychom mohli aplikaci pokud možno co nejlépe přizpůsobit potřebám studentů.

Následují tři části, každá pro jednotlivou metodu (algoritmus). Část pro tabu prohledávání je odlišná tím, že pro tento algoritmus žádná předchozí aplikace neexistovala. Z toho důvodu jsem navrhl dva způsoby vizualizace pro zhodnocení a možnost napsat, co by studenti dále považovali za vhodné.

V poslední části formuláře se nachází návrhy na obsah aplikace, které lze ohodnotit, a možnost připsat libovolné vlastní poznámky či nápady.

Pro formu odpovědí jsme vždy navrhli základní koncepty, se kterými lze pracovat, a nechali jsme je ohodnotit. Tento typ otázek „hodnocení možností“ měli respondenti ohodnotit na stupnici určitě ano, spíše ano, spíše ne, určitě ne. Pro hodnocení výsledků budu používat číselné ohodnocení 3, 2, 1, 0 (v pořadí určitě ano až určitě ne).

Tabulka 2.1: Výsledky možností z otázky 1

Pochopení jaký vliv mají jednotlivé parametry na průběh algoritmu	2,75
Vyzkoušení si použití iterativního algoritmu	2,56
Vyřešení problému danou iterativní metodou	2,25
Názorné vysvětlení látky z přednášek	2,22

2.1.2 Analýza výsledků

Formulář vyplnilo 32 studentů. Není to příliš vysoký počet, ale lze s tím rozumně pracovat. Myslím si, že za to může délka formuláře, nicméně se mi nepovedl více zjednodušit bez ztráty potenciálně užitečných odpovědí.

Otázka 1: Jak moc (by) přispěly následující možnosti k pochopení iterativních metod?

V tabulce 2.1 jsou sepsané odpovědi s průměrným výsledkem ze stupnice 0 (určitě ne) až 3 (určitě ano). Je zřejmé, že se respondentům všechny navrhované možnosti zdály vhodné. Jako nejvhodnější se jeví pochopení vlivu parametrů iterativní metody na průběh algoritmu. V podstatě jsem si potvrdil, že se prvotní představa základní funkcionality líbí.

Otázka 2: Máte ještě jiné zdroje pochopení iterativních metod?

V odpovědích se opakovalo nejvíce BI-ZUM (4) a youtube (3), potom jednotlivě přednášky, edux a náhodné články na webu.

Otázka 3: Při jaké příležitosti jste původní Java applety použil(a)?

U této otázky bylo možné zaškrtnout více odpovědí a také dopsat vlastní. Výsledky jsou shrnuté v tabulce 2.2. Většina respondentů použila applety na cvičení, to znamená, že byl přítomen cvičící a mohl jim práci s aplikací vysvětlit. Bohužel se téměř polovině respondentů nepodařilo applet spustit. To jen potvrzuje nutnost vytvoření aplikace, kterou lze v dnešní době spustit a používat.

Otázka 4: Na jakém zařízení jste aplikaci použil(a) nebo kde jste se ji pokusil(a) spustit?

Opět se jedná o otázku s více možnými odpověďmi. Výsledná data jsou následující:

- 65,6% - Vlastní počítač

Tabulka 2.2: Výsledky otázky 3

Na cvičení PAA týkajících se iterativních metod	19	59,4%
Nepodařilo se mi applet spustit	14	43,8%
Doma pro lepší pochopení látky během semestru	2	6,3%
Doma během přípravy na zkoušku	1	3,1%
Při vypracování 4. domácího úkolu	1	3,1%
Nepoužil	1	3,1%
Nepoužil, vizualizaci si pamatuji z BI-ZUM	1	3,1%
O appletech jsem vůbec nevěděl	1	3,1%

- 43,8% - Školní počítač
- 6,2% - Nepokoušel(a) jsem se

Převažuje možnost vlastní počítač. Nikdo však nezkoušel aplikaci spustit na mobilu, přestože byla dostupná přes web. Nicméně zde by se takový výsledek dal očekávat, jelikož problém s podporou Java appletů je již v běžných prohlížečích, natož v prohlížečích mobilních.

Otázka 5: Dáváte přednost webové nebo desktopové aplikaci?

- 87,5% - Webová aplikace
- 12,5% - Desktopová aplikace

Velmi konkrétní a zásadní otázka. Webové aplikace jsou moderní a mají v dnešní době spoustu výhod. Tím však desktopové aplikace nenahrazují ve všech odvětvích. Naši aplikaci máme na rozmezí, jelikož potřebujeme zpracovávat poměrně velký objem dat (ukládání historie běhů pro další zkoumání a porovnávání). Přitom je velmi důležitou součástí i jednoduchá dostupnost a použitelnost.

Výsledek je jasný, je ale potřeba zjistit jaké jsou důvody pro volbu daných odpovědí. Následuje otázka navazující na to, jakou možnost daný respondent zvolil.

Otázka 6.1: Proč dáváte přednost desktopové aplikaci?

Desktopové aplikace dávají přednost 4 respondenti, důvody jsou sepsány v tabulce 2.3. Všechny důvody souvisí s připojením, tedy až na to, že u webové aplikace musí mít webovou adresu. Některé obavy lze eliminovat i u webové

Tabulka 2.3: Výsledky otázky 6.1

Možnost práce offline	4	100%
Nehrozí přerušení spojení	4	100%
Nemusím si pamatovat webovou adresu	3	75%
Obávám se problému při nestabilním připojení	1	25%

Tabulka 2.4: Výsledky otázky 6.2

Nemusím nic stahovat	26	92,9%
Minimální požadavky na prostředí (webový prohlížeč)	24	85,7%
Nemám důvěru k cizím (studentským) programům	8	28,6%
Může být spustitelná i na mobilu	1	3,6%

Tabulka 2.5: Výsledky možností z otázky 1 k tabu prohledávání

Porovnání grafů průběhu stejné instance s jinými parametry	2,47
Graf hodnoty fitness funkce v průběhu algoritmu	2,39

aplikace. Webová aplikace se musí načíst, ale výpočet lze provádět lokálně bez nutnosti dotazování na server. Více se této problematice budu věnovat v sekci 2.3 (Návrh architektury).

Otázka 6.2: Proč dáváte přednost webové aplikaci?

Webové aplikaci dává přednost 28 respondentů, důvody jsou sepsány v tabulce 2.4. Všechny důvody jsou typické pro webové prostředí. Webový prohlížeč má dnes každý a jeho používání je tak běžné, že se spousta aplikací původně desktopových přesouvá na web.

Tabu 1: Odhadněte, zda přispěje k porozumění...

V tabulce 2.5 jsou odpovědi a průměrný výsledek ze stupnice 0 (určitě ne) až 3 (určitě ano). Oba návrhy jsou vítané, přednější se zdá být porovnávání běhu algoritmu na instanci s různým nastavením parametrů. Graf pro běh algoritmu se však bude hodit jako výborný indikátor běhu algoritmu. Uživatel tak nebude informován jen o tom, že algoritmus běží (například nějakým točícím se kolečkem), ale bude mít k dispozici doposud dosažené výsledky.

Tabulka 2.6: Výsledky možností z otázky 7

Vývoj ceny (graf)	2,86
Porovnání průběhu algoritmu na jedné instanci s různými parametry (2 až 4 datasety v jednom grafu)	2,65
Krokování algoritmu	2,55
Automatické/manuální opakování spouštění s různým seedem PRNG (pro randomizované algoritmy)	2,28
Nastavení/uložení seedu PRNG (pro randomizované algoritmy)	2,10

Tabu 2: Jakékoliv další nápady či poznámky?

Bohužel se jen jeden respondent odvážil napsat svůj názor. Jeho odpověď byla: „Vizualizace!“. Odpověď je velmi obecná, vizualizace můžou být i grafy z předešlé otázky.

Myslím si, že důvodem nedostatku nápadů od studentů je nízká popularita algoritmu tabu prohledávání v předmětu MI-PAA.

Otázka 7: Jak byste v aplikaci ocenili tyto možnosti?

Opět je k dispozici tabulka 2.6 s výsledky a hodnocení na stupnici 0 (určitě ne) až 3 (určitě ano). Potvrdilo se, že graf vývoje ceny a porovnání běhu s různými parametry je obecně žádané pro pochopení algoritmů. Krokování algoritmu bylo také hodnoceno velmi kladně. Zbylé dvě možnosti ohledně PRNG (pseudorandom number generator) se týkají především zbylých dvou algoritmů. Tabu prohledávání není založeno na generaci náhodných sousedů apod., nebudu zde proto zbylé možnosti uvádět.

Otázka 8: Co ještě byste rádi viděli v naší aplikaci?

Zde se rozepsali dva respondenti, následuje shrnutí požadavků na základě jejich komentářů.

- Jednoduché a intuitivní ovládání
- Další algoritmy
- Možné propojení s Algovision (MI-PAL)
- Tutoriál, který by provedl uživatele nastavením parametrů
- Možnost porovnat více algoritmů vzhledem k jedné instanci

Jednoduché a intuitivní ovládání by mělo být základem aplikace, určitě bude patřit do hlavních požadavků.

Co se týče dalších algoritmů, aplikace by měla být napsaná tak, aby přidání dalšího algoritmu nebyl problém, nicméně to není předmětem této práce. Navíc je další algoritmy v předmětu MI-PAA, pro který je aplikace primárně určena, nevyučují.

Návrh na propojení s Algovision zní zajímavě, nicméně po prozkoumání aplikace jsem dospěl k závěru, že se nejedná o hotovou a osvědčenou aplikaci, ze které by se dalo odrazit. Proto zůstáváme u tvorby aplikace nové.

Tutoriál je dle mého názoru dobrý nápad, ale do určité míry. Sám autor nápadu do poznámky uvedl, že si není jist, zda by to již nezastupovalo roli cvičícího nebo přednášejícího. Od aplikace bych očekával, že bude poskytovat informace o tom, co parametr ovlivní v rámci algoritmu. Jaký bude dopad parametru na průběh a výsledek algoritmu se však nelze jednoduše vysvětlit. Závisí totiž na problému i konkrétní instanci.

Zbývá možnost porovnat více algoritmů vzhledem k jedné instanci. Souhlasím s autorem nápadu, že by tato funkce mohla přinést zajímavé výsledky. Účel aplikace je trochu odlišný. Má sloužit k pochopení konkrétního algoritmu, konkrétní metody. Ať už při programování vlastní verze vybraného algoritmu nebo pro pochopení algoritmu ke zkoušce. Porovnání algoritmů mezi sebou by mohl být jakýsi sekundární cíl.

Určitě necháme relevantní nápady v sekci „nice to have“, ale snaha je implementovat hlavně klíčovou funkcionalitu v co nejlepší podobě.

2.2 Sestavení požadavků na aplikaci

Pro sestavení požadavků na aplikaci beru v úvahu analýzu výsledků formuláře, vlastní zkušenosti z předešlých projektů, konzultace s cvičícími a samozřejmě zadání práce.

Začnu se zadáním práce, které budu postupně rozšiřovat podle ostatních zdrojů.

Aplikace má mít přehledné a snadno použitelné grafické rozhraní a musí být spustitelná bez nutnosti instalace. Tento nefunkční požadavek podporuje i výsledek formuláře, kde převažuje podpora webové aplikace, která je spustitelná bez instalace. Dalším úkolem bude navržení grafického uživatelského rozhraní, kterému bude věnována celá kapitola a bylo také podpořeno respondenty.

Funkční požadavky ze zadání

- Možnost výběru z několika problémů k řešení
- Automatické i ruční generování instancí problémů

- Možnost práce s více instancemi a vracení se k nim
- Možnost nastavování parametrů Tabu prohledávání
- Možnost nastavování velikosti krátkodobé a dlouhodobé paměti (případně dalších parametrů)
- Sledování běhu algoritmu (graficky) s vytvářením statistik

Z formuláře dále vyplynulo, že k porozumění je vhodné porovnání běhu algoritmu na jedné instanci s různými parametry a případně různých instancí. Průběh algoritmu má být znázorněn grafem ukazujícím vývoj ceny stavů, které algoritmus prochází.

Uvedené požadavky již stačí k vytvoření splnění účelu aplikace, záleží ještě na zpracování uživatelského rozhraní.

Jelikož mám v úmyslu implementovat aplikaci se všemi základními požadavky tak, aby opravdu splňovali svůj účel, rozhodl jsem se sestavit rozumné minimum požadavků a zbylé označit jako vylepšení, která lze implementovat později. Aplikace tak bude funkční a nebude obsahovat spoustu možností, které použije jen pár studentů na úkor kvality jádra aplikace.

Další možná rozšíření jsou například krokování algoritmu, porovnání průběhu různých algoritmů nad jednou instancí a vizualizace vstupu. Krokování algoritmu se zdá jako velmi vhodné, nicméně u tohoto typu algoritmu naopak nejsou zásadní jednotlivé kroky. Jde o tendence běhu na základě parametrů, tedy jde o abstraktnější chování než pochopení daného kroku v konkrétní situaci. O konkrétní situaci jde v poměrně malém počtu případů a nemusí mít ani zásadní vliv na výsledek běhu. Proto je krokování algoritmu zařazeno do této kategorie. Dalším důvodem je rozdíl implementační složitosti. Návrh běhu algoritmu bez krokování bude logicky směřovat na samostatné vlákno pro výpočet, kvůli možnosti přerušení běhu a kontrole nad uživatelským rozhraním v době výpočtu. Krokování by tak nemělo znamenat příliš velkou změnu v logice existujícího kódu při pozdější implementaci oproti implementaci, která s krokováním počítá. Samotná implementace krokování obsahuje spoustu další práce a přeci jen jde i o časovou náročnost implementace.

2.2.1 Seznam požadavků na aplikaci

Funkční požadavky

- Volba algoritmu a problému (nezávisle na sobě)
- Správa instancí problému
  - Nahrát soubor instance
  - Vygenerovat instanci
  - Stažení souboru s instancí

  - Seznam nahraných instancí s výběrem instance pro výpočet
- Nastavitelné parametry zvoleného algoritmu (Tabu)
- Sledování běhu algoritmu na grafu
- Porovnání více běhů algoritmu
  - Na grafu
  - Tabulka s údaji běhu (čas běhu, cena řešení, nastavené parametry)

Ostatní (nefunkční) požadavky

- Spustitelnost bez instalace
- Jednoduché UI - minimalismus

2.3 Návrh architektury

Požadavky jsou sestavené a průzkum studentů vyhodnocený, nyní se zaměřím na to, kde a jak bude aplikace fungovat. Na jaké platformě a jaké bude mít požadavky na tuto platformu. Následuje vyhodnocení, zda má zvolená platforma potenciál pro splnění všech požadavků.

Jedná se o samostatnou aplikaci, která není závislá a ani na ní nezávisí jiná aplikace. Protože se má spustit bez instalace, nabízí se buďto spustitelný soubor nebo webová aplikace.

2.3.1 Porovnání možností desktopové a webové aplikace

Pokud by se jednalo o spustitelný soubor, další otázkou by bylo v jakém prostředí by byl spustitelný. V podstatě je na výběr Java aplikace, kterou lze spustit na většině operačních systémů nebo se omezit na jeden operační systém. Pokud by se jednalo o webovou aplikaci, multiplatformnost by zajišťoval webový prohlížeč a studenti nemusí nic stahovat.

Dalším aspektem jsou požadavky pro běh aplikace. Víme, že studenti aplikaci využívali primárně na cvičení MI-PAA, kde je k dispozici jak webový prohlížeč a připojení k internetu, tak Java. Webový prohlížeč a připojení má k dispozici v učebně každý i na svém zařízení, u Javy to není úplně jisté, ale lze předpokládat, že většina studentů fakulty informačních technologií má také nainstalovanou Javu.

U desktopové aplikace se respondentům líbilo, že se nemusí bát výpadku připojení, kvůli kterému by mohli přijít o výpočet. V dnešní době je však možné tomuto problému zabránit i u webové aplikace. Desktopová aplikace má stále výhodu v možnosti běhu offline, úplně bez připojení. Protiargumentem je opět dnešní doba, ve které máme možnosti připojení velmi rozsáhlé

a do budoucna se to bude jen zlepšovat. Nároky na připojení pro webovou aplikaci mohou být velmi nízké a pokud uživatel nepřijde o data při ztrátě připojení, webová aplikace se svou použitelností vyrovná desktopové i v těchto aspektech.

Webové aplikace mají ale také spoustu výhod, některé z nich potvrdili i respondenti. Ke spuštění stačí webová adresa, žádné stahování cizích aplikací a lze přizpůsobit i mobilním zařízením. Webové aplikace také nabízejí stále nové možnosti, jelikož je to uživateli velmi oblíbená platforma.

Výsledkem je tedy implementace webové aplikace, pokud se nenajde omezení, které by znemožňovalo splnění požadavků na aplikaci z předešlé sekce.

2.3.2 Návrh architektury webové aplikace

Základem architektury webových aplikací je obecně vždy server a klient. V tomto slova smyslu je server jen stroj, který vyřizuje příchozí požadavky na dané webové adrese. Za tímto strojem se však může skrývat další infrastruktura.

Co ale potřebuje aplikace pro demonstraci tabu prohledávání? Celá aplikace se specializuje na běh algoritmu. Jelikož by taková aplikace mohla být bez problému i desktopová, bude hlavní roli hrát klient. Server zde bude pro jednoduchost přístupu k aplikaci.

Pokud bychom chtěli aby výpočet běžel na serveru, nároky na takový server by byly velmi vysoké. Muselo by například existovat omezení maximálního počtu běžících výpočtů. Také by uživatel musel být periodicky informován o běhu algoritmu, jelikož má aplikace zobrazovat jeho průběh. Logicky lze tedy tuto možnost vyloučit. Výpočet musí probíhat u klienta, tím se tyto problémy odstraní. Toto řešení nabízí v podstatě téměř neomezenou škálovatelnost, jelikož si každý klient provádí výpočet sám.

Aplikace nevyžaduje ani žádnou společnou databázi, i data lze uchovávat lokálně. Navíc jsou data z aplikace povahově krátkodobá, slouží k pochopení dané tématiky a nemusíme je uchovávat dalších několik let. Proto není nutné zálohování či podobná opatření.

2.3.2.1 Cílové postředí aplikace

Zbývá poslední otázka ohledně architektury aplikace. Lze splnit všechny požadavky v prostředí webového prohlížeče a to převážně na straně klienta?

Mám v úmyslu využít moderních technologií a to převážně HTML5, CSS3 a JavaScript.

Když se podíváme na dnešní webové stránky či aplikace, převážná většina používá nějakým způsobem JavaScript. Jeho podpora a možnosti za poslední roky velmi vzrostly a pravděpodobně ještě růst budou. Výjimku tvoří například hry nebo multimediální přehrávače, které používají Flash. Dříve používané Java applety již nejsou podporované vůbec a Flash se už využívá jen na specifické typy aplikací.

Všechny běžně používané prohlížeče HTML5, CSS3 a JS podporují. V dnešní době je další výhodou pro vývojáře, že se prohlížeče aktualizují automaticky, takže odpadá nutnost zpětné kompatibility pro soustu verzí.

Když se podíváme na uživatelské preference webových prohlížečů [8], zjistíme, že skoro 60% uživatelů používá Chrome. Pro tuto aplikaci máme však ještě užší spektrum uživatelů a téměř bychom se mohli spolehnout čistě na Chrome, který je k dispozici i na všech školních počítačích.

Vrátím se ale k původní otázce, lze splnit požadavky v tomto prostředí, tedy při použití HTML5, CSS3 a JS?

Potřebujeme na straně klienta spouštět delší výpočet, takže na výpočetním vlákně, odděleném od UI. To umožňuje API Web Workers (HTML Living Standard [9]). Zároveň lze v průběhu výpočtu mezi vlákny komunikovat, takže je možné zobrazovat průběh algoritmu.

Dále potřebujeme uchovávat vstupní data (instance problémů) a výstupní data (výsledky algoritmu). Vstupní data lze generovat nebo nahrávat soubory. V JavaScriptu lze číst uživatelem vybrané soubory přímo z disku. Na uložení referencí na vybrané soubory a na uložení vygenerovaných instancí a výstupních dat je k dispozici localStorage. LocalStorage má ale omezenou paměť (podle prohlížeče) a tak je vhodnější použít IndexedDB, což je implementace key-value databáze v prohlížeči. Výhodou je možnost základní strukturalizace a ukládání většího množství dat.

Stažení souboru instance (ať už přidaného uživatelem nebo vygenerovaného) lze zajistit HTML tagem *a*.

Posledním problémem by mohla být vizualizace běhu, výsledků a následné porovnávání pomocí grafů. Naštěstí existuje spousta knihoven pro vytváření grafů pomocí JavaScriptu. V této aplikaci bude jedním z klíčových vlastností rychlost vykreslování. Optimalizaci vykreslování grafů budu popisovat v samostatné sekci.

Všechny ostatní požadavky lze splnit pomocí vybraných technologií, důležitá bude konkrétní implementace.

Požadavky na klienta jsou vymezeny, zbývají požadavky na server a ty jsou velmi malé. Stačí mít přístupné soubory pro klienta, jelikož si je klient načte a zbytek práce server nijak neovlivňuje.

Kapitola 3

Návrh uživatelského rozhraní

Značná část návrhu uživatelského rozhraní probíhala v rámci předmětu MI-NUR. Šlo o postupné zpracování částí návrhu prokládané konzultacemi.

V první části se jednalo o rozvahu a vytvoření wireframe (lo-fi prototypu). Následovala analýza podobných aplikací a vytvoření prototypu v cílové platformě (hi-fi prototypu). Nakonec bylo samostatné uživatelské rozhraní testováno a následně upraveno podle výsledků testů.

3.1 Rozvaha a model aplikace

Tato část se zabývá prvotním uchopením aplikace. Slouží k ucelení představy jak bude vypadat a co bude obsahovat. Jedná se především o hlavní cíle, od kterých se bude odvíjet zbytek aplikace.

3.1.1 Product statement

Aby bylo možné začít uvažovat o budoucím vzhledu aplikace, musí se stanovit platforma, pro kterou je aplikace určena. Z analýzy již vyplynulo, že se bude jednat o *webovou aplikaci*. Dále je dobré mít *product statement*, který bude v krátkosti vystihovat účel aplikace, aby se o ní lépe mluvilo.

> Projekt slouží k demonstraci běhu iterativních algoritmů. Bude sloužit studentům k lepšímu pochopení těchto algoritmů v předmětu MI-PAA za pomoci grafického zobrazení průběhu výpočtu.

3.1.2 Business requirements

Dále se stanovují byznys požadavky (business requirements), jedná se o shrnutí požadavků z hlediska klíčových hodnot, které má aplikace poskytovat.

- Grafická prezentace běhu algoritmů

- Řešení několika druhů problémů
- Generátor náhodných instancí problémů
- Řešení problémů pomocí algoritmů:
  - Simulované ochlazování
  - Genetický algoritmus
  - Tabu prohledávání
- Možnost nastavení parametrů daného algoritmu

3.1.3 Použití uživatelského rozhraní

Dalším bodem jsou případy užití (use cases), jedná se ale o *případy užití z hlediska návrhu UI*. Jde o strohou představu hlavních sekcí aplikace. Z těchto bodů se vychází při tvorbě wireframů.

- Na hlavní stránce nalezneme:
  - Taby s jednotlivými algoritmy
- Na stránce s algoritmy nalezneme:
  - Panel s parametry
  - Boční panel s výběrem instancí
  - Prostor, kde je možné zobrazit:
    - Graf zobrazující vývoj nejlepšího řešení
    - Graf zobrazující průchod stavovým prostorem
    - Stránku s výsledky
  - Pravý panel s historií již řešených problémů
  - Na bočním panelu nalezneme tlačítko umožňující přejít do generátoru nebo nahrání problému ze souboru
- Na stránce generátoru nalezneme:
  - Volba problému generované instance
  - Nastavení parametrů vygenerované instance

3.1.4 Task list

Seznam akcí nebo úkonů (task list) shrnuje, jaké možnosti interakce s daným logickým celkem chceme uživateli poskytnout. Tyto úkony jsou v podobě grafu na obrázku 3.1.

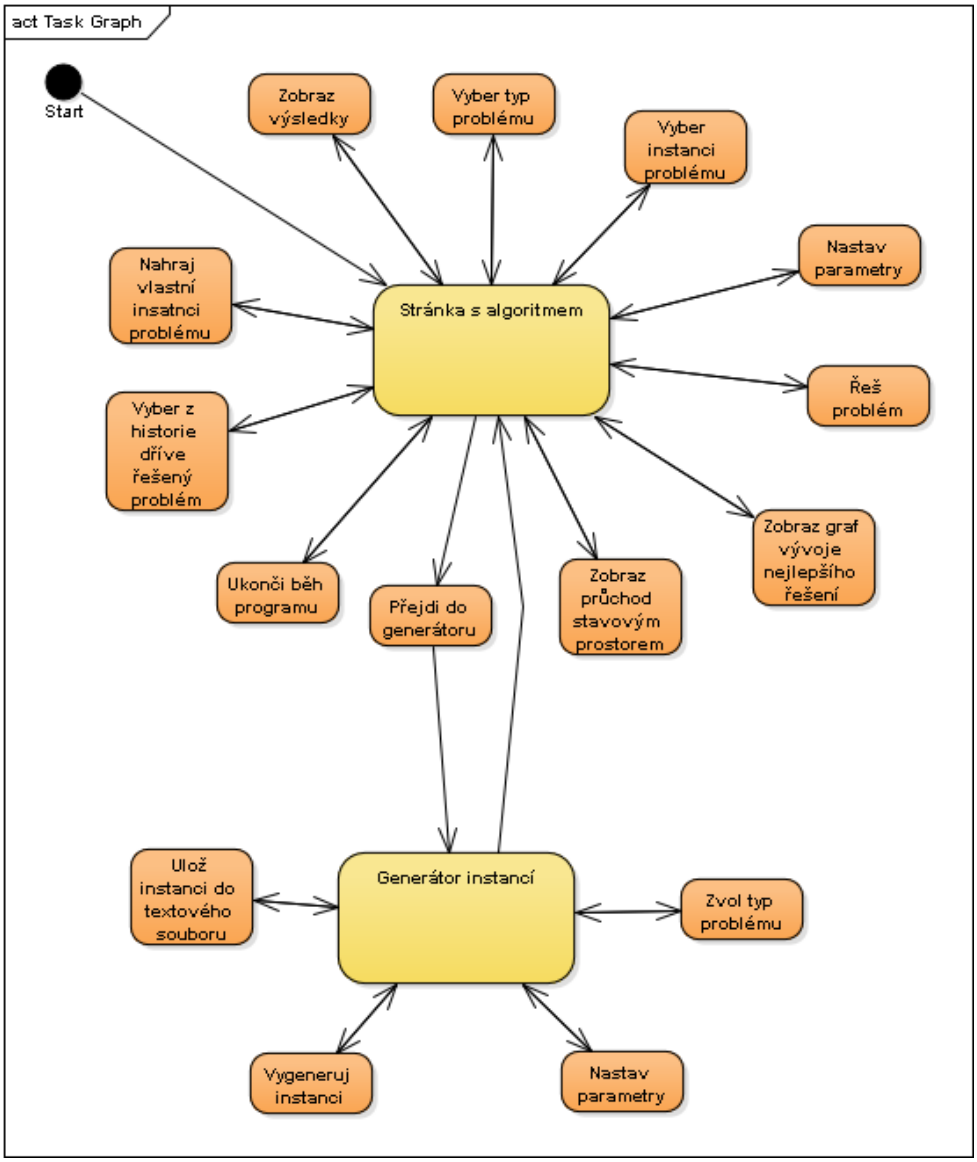
- Algoritmy
  - Vyber typ problému
  - Vyber instanci problému
  - Nastav parametry
  - Řeš problém
  - Zobraz graf vývoje nejlepšího řešení
  - Zobraz průchod stavovým prostorem
  - Zobraz výsledky
  - Ukonči běh programu
  - Vyber z historie dříve řešený problém
  - Nahraj vlastní instanci problém
  - Přejdi do generátoru
- Generátor
  - Zvol typ problému
  - Nastav parametry
  - Vygeneruj instanci
  - Ulož instanci do textového souboru

3.1.5 Wireframe

Pro prvotní návrh UI zbývá vytvořit lo-fi (low-fidelity) prototyp v podobě wireframe modelu. Ten slouží k rychlému rozvržení struktury vzhledu aplikace. Výhodou je rychlost jeho vytvoření a úprav. Slouží také k umožnění diskuze nad konkrétním vizualizovaným konceptem [10].

Celé UI aplikace je v anglickém jazyce. Výhodou je, že aplikaci mohou používat studenti jak českého tak anglického programu. Čeští studenti však nejsou opomenuti. Magisterské studium již předpokládá určitou úroveň anglického jazyka. Některé předměty mají i všechny studijní materiály v angličtině. Naše aplikace navíc bude obsahovat převážně termíny a případně stručný popis. Studenti by s tím neměli mít žádný problém a odpadá nutnost překladu.

Tento prototyp představuje první vizuální podobu, proto jsme spolupracovali při rozvržení logiky vzhledu. Michal Kluzáček následně v rámci své práce [1] zrealizoval vytvoření wireframe prototypu. Já a Adam Kugler jsme přispěli připomínkami, které se při shodě názorů zapracovaly.



Obrázek 3.1: Task graph

3.2 Průzkum podobných aplikací

Průzkum se provádí záměrně až po prvotním návrhu (lo-fi prototypu), aby se nezarazila mez fantazii. Nicméně samotná fantazie nestačí. Účelem průzkumu je získání a porovnání nápadů, nalezení používaných řešení ovládacích prvků, které uživatelům přijde intuitivní.

Za normálních okolností by proběhla analýza aktuálních aplikací, které se budou nahrazovat. Bohužel je velký problém stávající Java applety vůbec spustit, jelikož aktuální verze prohlížečů je již vůbec nepodporují a po stažení starších verzí se automaticky provede aktualizace.

Pro účely návrhu UI se lze spokojit i s aplikacemi, které nabízejí typově podobné UI. Typově souhlasí většinou webové stránky, které jsou zaměřeny na simulaci nebo ukázku nějakého algoritmu.

Průzkum realizoval Adam Kugler v rámci své práce [2].

3.3 Vytvoření prototypu aplikace pro UI testování

Vytvoření hi-fi prototypu v cílové platformě, jsem realizoval já. Vycházel jsem z lo-fi prototypu a inspiroval se běžně používanými webovými prvky. Pro základ stylování jsem použil *bootstrap* a později jsem přidal také knihovnu ikon *font-awesome*.

Hlavním ovládacím prvkem jsou záložky (taby) sloužící k výběru algoritmu. Výběr algoritmu je pro aplikaci hlavní. Předmět MI-PAA se zabývá postupně jednotlivými algoritmy. Cílem je pochopení každého algoritmu z pokročilých iterativních metod, nikoliv jejich porovnávání. Rozložení aplikace zachycuje obrázek 3.2.

Zbytek plochy webového prohlížeče je rozdělen na tři sloupce. Levý panel (obrázek 3.4) sdružuje vstupní data a klíčový prvek aplikace - tlačítko start. Pravý panel (obrázek 3.4) obsahuje historii výpočtu, tedy výstupní data. Prostor uprostřed je značně větší a slouží zobrazování aktuálních informací a vizualizaci pomocí grafu. Aktuální informací je buďto aktuální stav výpočtu včetně grafu s dosavadním průběhem nebo výsledky dokončených běhů.

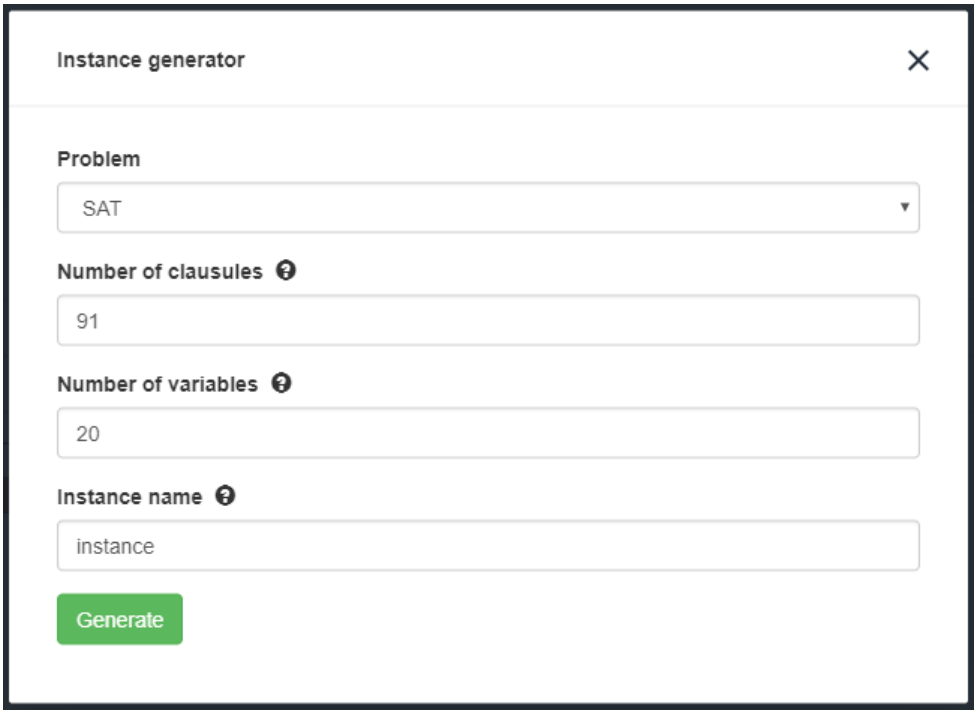
Jediná část, která se zobrazí na kliknutí je generátor (obrázek 3.3). Ten se zobrazí v modal okně a obsahuje výběr problému a parametrů instance.

Logika tedy koresponduje se směrem čtení nebo prohlížení aplikace. Lidé běžně prohlíží stránku shora dolu a zleva doprava.

Tak je koncipován i návrh UI/UX. Nahoře uživatel vybere algoritmus, následně vlevo nastaví parametry (pro algoritmus, vybere problém a jeho instanci) a klikne na tlačítko „start“. Vlevo řeší vstup uprostřed je vidět průběh a vpravo se shromažďuje historie běhů algoritmu (výstup).

Při první návštěvě má uživatel předvyplněny všechny parametry, aby mohl jen zmáčknout tlačítko „start“ a až později řešit detaily. Tím uživatele hned neznechutíme vyplňováním sousty povinných polí, aby zjistil, co aplikace dělá

Obrázek 3.2: Náhled rozložení aplikace - prototyp



Obrázek 3.3: Generátor instancí - prototyp

nebo umí. Vše je okamžitě připraveno a postupně může začít s procházením aplikace, podle toho, co každého uživatele zaujme.

Celkově je aplikace koncipována jako *single page application* (SPA). Vše je interaktivní, algoritmy sdílí problémy a jejich instance a každý algoritmus má zapamatované nastavení, takže je vždy možnost vybírat mezi problémy a algoritmy bez ztráty vyplněných údajů.

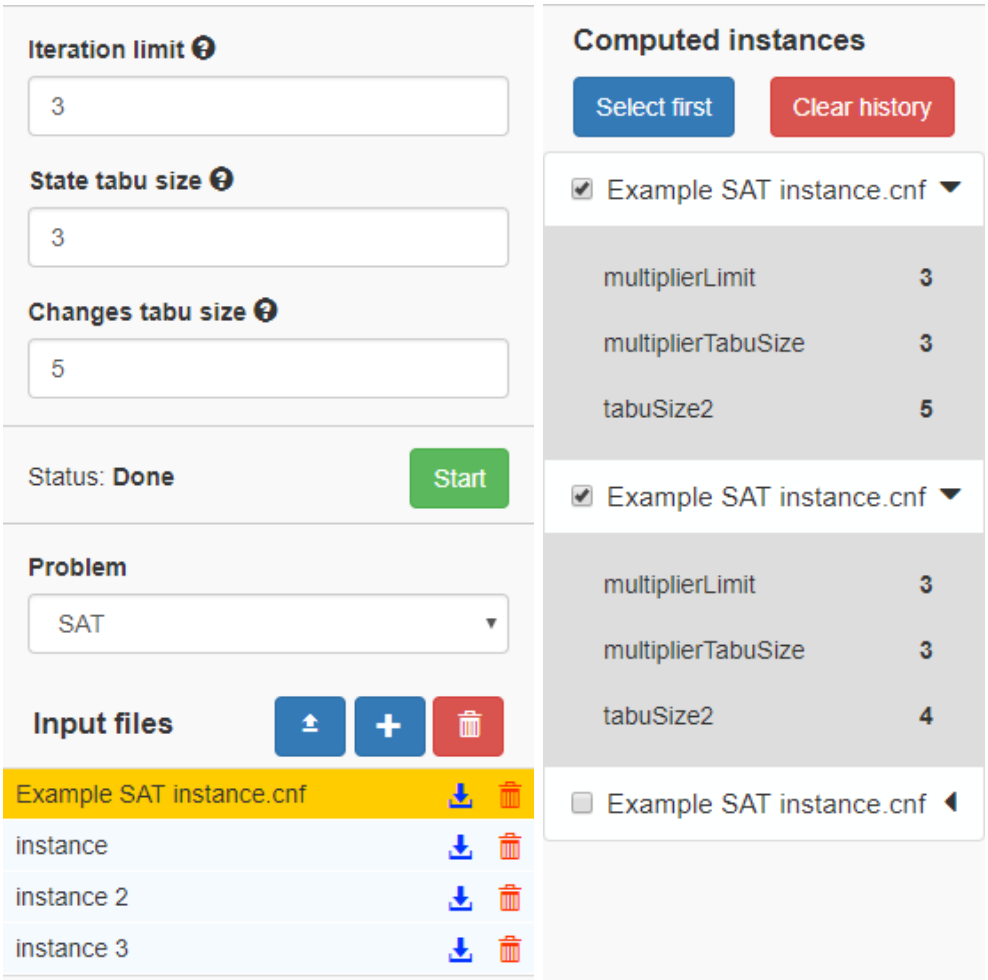
Každý panel má vlastní scrollbar a celá stránka tím pádem scrollbar nemá. Uživatel tak může nezávisle na pozici prohlížení výsledku procházet historii nebo například vstupní instance.

3.4 Testování UI

Prototyp je vytvořen a následuje testování. Nejprve byla provedena *heuristická analýza* a následně *uživatelské testování*. Jakožto tvůrce prototypu jsem nalezené chyby také opravoval.

3.4.1 Heuristická analýza

Heuristickou analýzu provedl Adam Kugler [2] podle pravidel Jakoba Nielsena. Každý problém ohodnotil na stupnici 1 (málo závažný) až 5 (velmi závažný).



Obrázek 3.4: Levý a pravý panel před testováním prototypu

Zde jsou ve stručnosti body, u kterých byl nalezen problém. Čísla v závorkách udávají hodnotu závažnosti.

- Shoda mezi systémem a realitou

  (4) Výpis názvů proměnných místo popisu parametrů v historii

  (3) Nejasný význam nadpisu „Computed instances“

  (2) Nejasný nadpis „Input files“ (neodpovídá reálnému obsahu)

- Minimální zodpovědnost (a stres)

  (2) Nelze smazat jen jednu položku historie

- Prevence chyb
  - **(2)** Možnost zadání chybných parametrů
  - **(4)** Lze kliknout na tlačítko „start“ s chybnými parametry
- Estetika a minimalizmus
  - **(1)** Nadbytečná informace při potvrzení akce v modálním okně (nadpis „Are you sure?“)
- Nápověda a dokumentace
  - **(4)** V aplikaci chybí popis očekávaných/akceptovaných formátů instancí

3.4.2 Uživatelské testování prototypu

U uživatelského testování prototypu jsme byli přítomni všichni, nicméně zdokumentování a vyhodnocení provedl Adam Kugler [2].

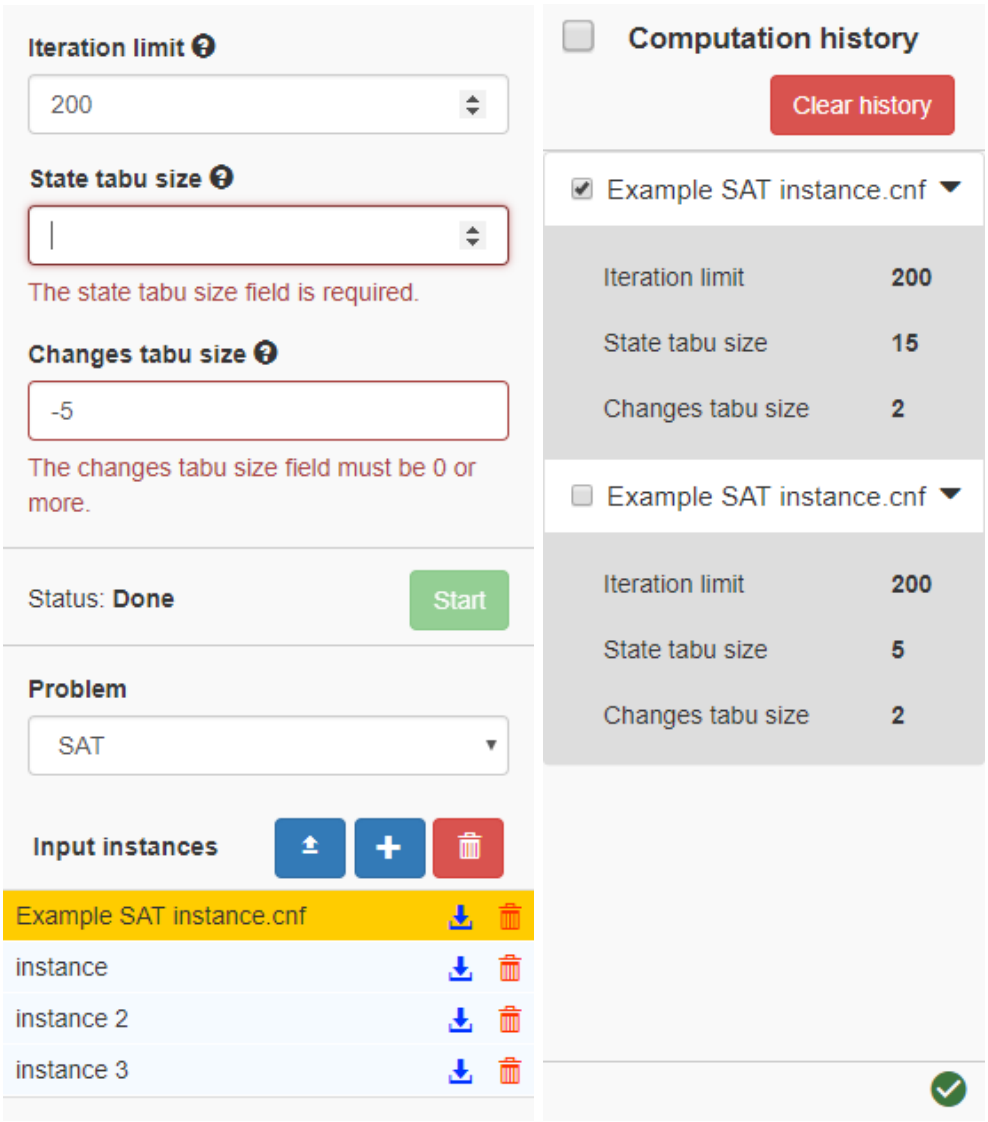
U porovnávání historie bylo tlačítko „Select first“, kterým šlo vybrat první instanci, což je stav po doběhnutí výpočtu instance. Toto chování se jevilo intuitivní jen po doběhnutí instance, nikoliv jakožto akce, kterou by uživatel chtěl později vyvolat. Závažnost problému má hodnotu 4 a řešením bylo odstranění tlačítka a přidání checkboxu, který vybíral a rušil výběr všech instancí jako je tomu u jiných běžně používaných aplikací (např. gmail).

Ze souborů nebylo možné určit parametry instance (identifikace jen podle názvu souboru). Závažnost 3, řešením bylo zobrazení parametrů po najetí myší na ikonku „info“ u libovolné instance.

Dále šlo v prototypu nastavit nevalidní vstupní parametry. Závažnost 5, jelikož uživatel neměl tušení o nevaliditě až do spuštění výpočtu. Řešením bylo přidání interaktivní validace. Zobrazení po jedné chybové hlášce u pole, které má chybnou hodnotu a červené podbarvení.

Posledním nalezeným závažným problémem byla neintuitivní funkce tlačítka smazat historii. Akce smazala historii napříč problémy a nikoliv jen historii aktuálně zobrazovanou v seznamu. Závažnost 3 a problém byl vyřešen opravou této funkce.

Opravy problémů byly poměrně jednoznačné. Jednalo se především o validaci vstupních parametrů a opravu neintuitivních ovládacích prvků. Úpravy jsou znázorněny na obrázku 3.5. Formulář je v nevalidním stavu a tak je tlačítko „start“ neaktivní. V pravém panelu byla také přidána ikonka indikující podporu zvolených technologií. Kdyby uživatel aplikaci spustil v prohlížeči, který nějaké z použitých rozhraní nepodporuje, objeví se červený křížek s vysvětlením a popisem chyby.



Obrázek 3.5: Levý a pravý panel po úpravě prototypu

Kapitola 4

Nástroje pro vývoj a spolupráci

Jedním z prvních kroků již při vytváření prototypu bylo založení *git* repozitáře. Výhody verzování není třeba nijak rozepisovat.

Kromě základních operací z kategorie `git pull` a `git push` jsme ještě použili například `git tag`. Ten jsme použili pro označení jednotlivých fází vývoje jako prototyp před a po testování a funkční aplikace před a po testování. Díky tagu jsme se mohli kdykoliv rychle vrátit na starší verzi a porovnat změny nebo pořídit snímky pro naše diplomové práce.

4.1 Github nástroje

Pro verzování jsme zvolili konkrétně github. Důvodem je využití nástrojů github, konkrétně *issues*, *projects*, *wiki* a *github pages*.

Nejprve jsme používali project pro shromažďování nápadu a poznámek na vylepšení. Github project umožňuje vytvoření sloupců jakožto kategorií a následné přidávání poznámek do jednotlivých sloupců a přesouvání mezi sloupci. Tím jsme si setřídily úkoly a nápady, abychom postupovali systematicky a na nic nezapomněli.

V pozdější fázi, kdy už jsme věděli, co přesně potřebujeme udělat nebo někdo našel chybu, jsme využili *issues*. Github issues je v podstatě jeden z bug trackerů. Lze ale použít i pro rozdělení úkolů. Využili jsme možnost přidávat značky (labels), milníky (milestones) a řešitele k jednotlivým úkolům/problémům (issues).

Dále jsem vytvořil dvě stránky na wiki, kde jsem popsal základní použité technologie s odkazy na dokumentaci a postup nasazení.

Pro nasazování testovací verze jsme použili *github pages*. Pro zjednodušení nasazení jsem vytvořil build task přímo pro tento účel. Nasazení na github pages je jednoduché, zvláště pro frontendovou aplikaci. Stačí nahrát do repozitáře do složky `docs` sestavené („zbuilděné“) soubory. Výsledek se objeví na url přiřazené k repozitáři.

4.2 Nástroje pro vývoj

Vývoj aplikace v jazyce JavaScript vyžaduje pouze libovolný textový editor, *node.js* a příkazovou řádku.

Hlavním bodem je *node.js*. Jedná se o softwarový systém pro psaní aplikací v jazyce JavaScript. Jeho součástí je největší balíčkový ekosystém volně šiřitelných knihoven - *npm*. Tento systém spravuje závislosti aplikace. Díky tomu má vývojář jednoduchý přístup ke spoustě užitečných knihoven.

Kromě samotného balíčkového systému, který je základem veškerého externího kódu aplikace, lze pomocí *node.js* spustit server pro lokální vývoj a sestavit optimalizovaný zdrojový kód pro produkční prostředí.

4.3 Nástroje pro debugging

Jelikož se jedná o webovou aplikaci, hlavním testovacím a debugovacím nástrojem je *Google Chrome*. *Chrome Developer Tools* je sada nástrojů zabudovaná do prohlížeče *Chrome*. Použil jsem hlavně konzoli, průzkumník elementů (HTML a CSS), síťovou komunikaci a zobrazení obsahu localStorage a IndexedDB.

Speciálně pro použitý framework je vytvořen addon *Vue Dev Tools*. Ten umožňuje procházet komponenty, zobrazit hodnoty aktuálních proměnných a případně i měnit jejich hodnoty. V rámci tohoto nástroje je také možné procházet stavový strom (vuex - více v kapitole implementace) nebo zobrazit a procházet historii změn stavu.

Kapitola 5

Implementace

5.1 Použité frameworky a knihovny

5.1.1 Vue.js

Základem aplikace je framework *Vue* [11]. Je poměrně jednoduchý na naučení a právě v tom je jeho síla. Stavební jednotkou je komponenta. Komponenty lze velmi jednoduše a rychle vytvořit. Díky tomu je aplikace rozdělena do menších částí, které lze rychle a přehledně spravovat. Také se dá velmi dobře přizpůsobit k nejrůznějším účelům.

Aplikace se sestavuje pomocí nástroje *Webpack*. Ten umožňuje vývojářům například hot module reload (hmr), při změně kódu se v aplikaci zamění pouze část, která byla upravena a to živě, bez obnovení stránky v prohlížeči. Tento efekt značně urychluje a zjednodušuje vývoj aplikace. Webpack také umožňuje používat nový způsob importu a exportu modulů, který v prohlížečích ještě není nativně úplně implementován (aktuálně - ES2015 - přidáno do standardu jako „initial definition“).

5.1.2 Vuex

Vuex [12] je jedna ze základních knihoven Vue. Poskytuje správu stavu celé aplikace pomocí „single state tree“, stromu, který je rozdělen podle stavových modulů (a nezávisí na hierarchii komponent). Každý modul má svůj stav, gettery, mutace a akce. Pro aplikaci se Vuex hodí, jelikož pracuje s poměrně velkým množstvím dat. Tato data sdílí více komponent a pomocí této správy stavu jsou data organizovaná, uložená na jednom místě a dostupná všude, kde je potřeba.

5.1.3 Worker loader

Worker loader je modul umožňující jednoduše vytvořit instanci třídy `Worker` (HTML5 Web Worker [9]) v aplikaci sestavované pomocí softwaru Webpack.

HTML5 Web Worker je velmi důležitá část aplikace. Javascript běží pouze jednovláknově a to by znamenalo, že při průběhu algoritmu nebude možné ovládat UI aplikace. Vytvořením instance třídy `Worker` si JS alokuje také výpočetní vlákno. Jedná se zatím o jediný způsob jak v JS lze provádět operace nad více vlákny.

Správně by se v dnešní době měla jakákoliv výpočetně náročnější akce na straně klienta provádět právě přes toto rozhraní. Důvod je prostý, uživatel má mít kontrolu nad UI, nikoliv čekat půl sekundy nebo i několik sekund než se dokončí operace. Naopak má být informován o tom, že taková operace probíhá a mít možnost operaci přerušit. A právě tyto výhody aplikace podporuje díky modulu worker loader.

5.1.4 Zangodb

Knihovna *zangodb* implementuje dotazovací jazyk podobný MongoDB [13]. Hlavním rozdílem je, že interně využívá HTML5 IndexedDB. IndexedDB je jednoduchá key-value databáze implementovaná v moderních prohlížečích.

Původní návrh pracoval s knihovnou IDB. Ta obalovala IndexedDB a místo zpětných volání (callback) se pracovalo s rozhraním třídy `Promise` (reprezentuje možné dokončení/chybu asynchronní operace). Důvodem ke změně byla špatná podpora prohlížečů a složitější používání knihovny. Zangodb je velmi přímočará a poskytuje i pokročilejší konstrukce dotazování.

Co se týče důvodu využití, tak vzhledem k velikosti vstupních a výstupních dat není vhodné použít localStorage, protože má většinou omezenou velikost (okolo 6 MB). Nechat všechna data pouze v paměti také není praktické, protože by uživatel při obnovení stránky nebo zavření okna/záložky prohlížeče o všechna data přišel.

5.1.5 Dc.js

DC je zkratka pro dimensional charting [14]. Knihovna využívá další dvě knihovny: crossfilter a d3. Umožňuje renderování data-driven reaktivních grafů. Crossfilter umožňuje definovat dimenze nad daty, která lze přidávat či odebírat, přičemž jsou tyto operace optimalizovány a jsou výrazně efektivnější než zpracovávání celé kolekce dat. Dimenze definovány pomocí crossfilteru se přiřadí k grafu, který je renderován pomocí d3.

Díky této knihovně může aplikace uživatelům vizualizovat průběh, výsledky a jakákoliv jiná zajímavá data.

5.1.6 Bootstrap a uiv

Bootstrap je velmi známý framework zaměřený na HTML s CSS, používá se jako základ pro jednotný vizuální styl [15]. Poskytuje také některé komponenty, ze kterých byl použit například *popover*. Komponenty jsou však závislé

i na JS a základní bootstrap JS by mohl kolidovat s logikou frameworku (Vue). Proto byla použita knihovna *uiv* [16], která bootstrap komponenty implementuje ve *Vue*. Výhodou je snazší použití a hlavně nekolidující kód v JS, tedy funkčnost v rámci *Vue*.

5.1.7 Další použité knihovny

Mezi další knihovny, které stojí za zmínku patří *vee-validate*, *sweet-modal-vue*, *vue-localStorage* a *vue-resource*.

Vee-validate umožňuje validovat formuláře pomocí nastavení atributů udávající omezení na vstup u daného formulářového prvku. K chybám u validace daného prvku lze jednoduše přistoupit pomocí zavolání funkce `errors` s parametrem obsahujícím název formulářového prvku. Poskytuje také předgenerované chybové hlášky, do kterých lze doplnit vlastní pojmenování pole, aby se nezobrazovala hodnota z kódu.

Sweet-modal-vue je jednoúčelová knihovna pro práci s modálními okny ve *Vue*.

Pro jednoduchou práci s *localStorage* je použita knihovna *vue-localStorage*. Hlavním využitím je uložení aktuálního stavu aplikace jako je výběr algoritmu a problému nebo aktuálně porovnávaných výsledků z historie. Uživatel tak může přepínat mezi algoritmy a problémy, neztratí stav porovnávání a po obnovení stránky bude tam, kde skončil.

Aplikace sice nevyužívá žádné API ze severu, ale načítá soubor výchozí instance pro každý problém, pokud uživatel při příchodu na stránku žádné instance nemá. Knihovna *vue-resource* toto načítání umožňuje.

5.2 Kompozice použitých technologií

Prvotní obsah projektu jsem vygeneroval pomocí *vue-cli*. Jelikož jsme chtěli používat výhody softwaru Webpack, zvolili jsme příkaz:

```
$ vue init webpack <název-projektu>
```

Tím se vytvořil seed projektu, včetně souborů pro *git*, konfigurace pro lokální vývojový server a pro sestavení produkčního balíčku.

Následovaly drobné úpravy konfigurace a s postupem času se přidávaly vybrané knihovny. Vybrání každé knihovny předcházel průzkum a většinou také testování různých možností, abychom našli funkcionalitu, kterou potřebujeme.

5.2.1 Vue

Jak už bylo zmíněno, framework *Vue* byl základem aplikace. Tento framework lze použít například také jen na specifický prvek v HTML a není nutné použít kompozici komponent. Pro náš účel je ale vhodnější rozdělení logiky do komponent, kdy jedna komponenta odpovídá jednomu souboru.

Využili jsme typu souboru `.vue`, jehož obsah definuje vždy právě jednu komponentu. V rámci souboru lze definovat šablonu (template), logiku komponenty (v JS) a styl komponenty. Celá aplikace používá JS *ES2015*.

Šablona obsahuje HTML obohacené o vue prvky. Lze například přistupovat k proměnným a metodám v rámci komponenty, používat podmínky a cykly, zachytávat události jako kliknutí myši nebo vkládat další komponenty.

Logika komponenty je reprezentována objektem v JS. Tento objekt má určené rozhraní, pomocí kterého se definuje logika komponenty. Například atribut `data` definuje funkci, která vrací výchozí data, se kterými se v rámci komponenty pracuje. Atribut `methods` definuje objekt s metodami, které lze v rámci komponenty používat a to buď v JS nebo je lze přiřadit k události v šabloně.

Styl komponenty se definuje pomocí CSS, lze použít i libovolný CSS preprocesor, pokud máme daný preprocesor ve Webpack konfiguraci. My jsme použili *Sass* (formát *scss*). *Vue* komponenta má ještě výhodu v tom, že u stylu můžeme zvolit možnost `scoped`, čímž docílíme, že všechny styly definované v rámci komponenty (resp. tagu styles s atributem scoped) budou aplikovány pouze na elementy z šablony této komponenty.

Pro představu jak vypadá celá komponenta je kód velmi jednoduché komponenty v ukázce 2.

V šabloně obsahuje především výpis proměnné pomocí dvojitých složených závorek a registraci zavolání funkce `reverseMessage` při kliknutí na tlačítko.

V JS export objektu komponenty, definici výchozích dat a jedné metody.

Styl má atribut `scoped`, takže se aplikuje jen na elementy `p` v rámci šablony komponenty.

Na obrázku 5.1 je znázorněno rozdělení logiky do jednotlivých komponent. Ne všechny komponenty jsou zobrazené najednou a pokud jsou označené jako specifické, pak jejich obsah závisí na aktuálním stavu.

5.2.2 Vuex

Vue komponenty jsou skvělý nástroj na oddělování logiky jednotlivých částí aplikace. Jsou ale situace, kdy potřebujeme pracovat s daty, která ovlivňují více komponent. Zde je několik možností jak s takovými daty pracovat.

První možností je předávat data z rodičovské komponenty do vnořené komponenty. V rodičovské komponentě lze také odchytávat události z vnořené komponenty. Tato komunikace je poměrně jednoduchá na pochopení, ale má své nevýhody. Každá změna dat znamená rozeslání události všemi směry, kde s daty chceme pracovat. To znamená množství kódu starající se o to, aby byla data synchronizována po každé úpravě. Další nevýhodou je komunikace s komponenty, které jsou ve stromu elementů sourozenci. Taková komunikace musí proběhnout přes rodičovské komponenty a to i v případě, že rodičovská komponenta s daty nepracuje (slouží pouze jako zprostředkovatel komunikace).

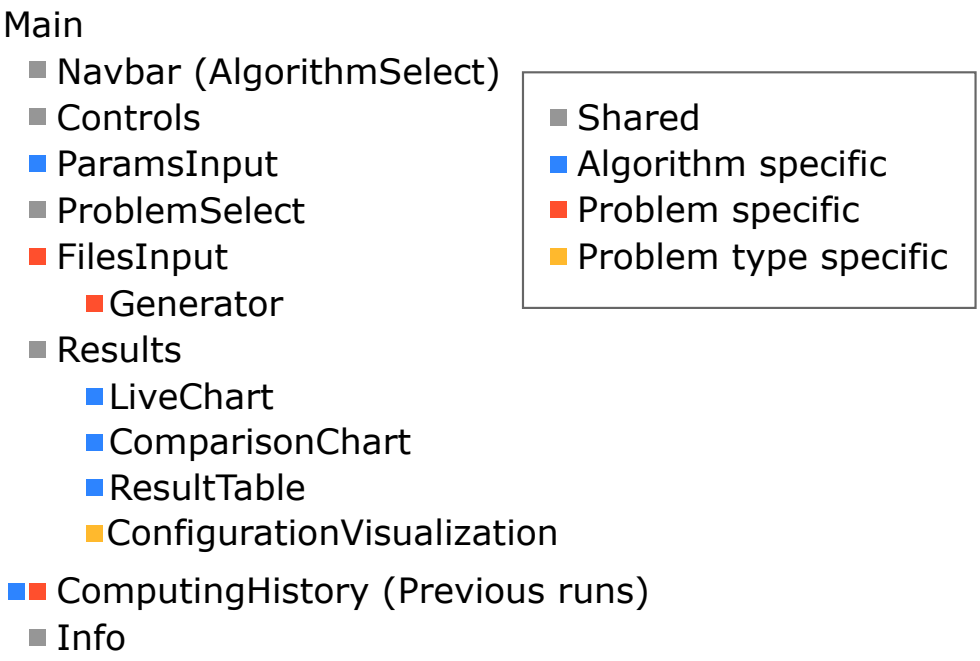
```
<template>
    <div class="example-component">
        <p>{{ message }}</p>
        <button v-on:click="reverseMessage">
            Reverse Message
        </button>
    </div>
</template>

<script>
export default { // JS object
    data() {
        return { message: 'Hello Vue.js!'; }
    },
    methods: {
        reverseMessage() {
            this.message = this.message.split('')
                .reverse().join('');
        }
    }
}
</script>

<style lang="scss" scoped>
    p { color: red; }
</style>
```

Zdrojový kód 2: Ukázka komponenty (souboru typu vue)

Tento typ komunikace slouží v podstatě pro základní výměnu dat, ideálně jedním směrem a do jedné komponenty.

Druhou možností je komunikace přes sběrnici událostí. Oproti prvnímu způsobu je komunikace nezávislá na rozmístění komponent. Nevýhodou je nepřehlednost, pokud takovou sběrnici používáme pro různé operace. Každá událost má svůj identifikátor (většinou řetězec), na který lze poslat zprávu a odposlouchávat zprávy příchozí. Párovat větší množství dat je i pro tento způsob obtížné a hlavně nepřehledné.

Třetí možností je správa stavu pomocí *vuex*. Nevýhodou je počáteční práce s vytvořením struktury pro uchovávání stavu, proto se pro malé množství sdílených dat se obvykle nevyužívá. Výhodou je přehledná struktura definovaná na jednom místě s možností rozdělení do modulů. Typicky se používá, pokud více komponent sdílí jeden stav (přesněji část dat) nebo pokud různé akce napříč aplikací mění nějakou část sdílených dat. Naše aplikace má všechna data uložena v sobě, případně v lokální databázi prohlížeče, využívá tak obě typické výhody této knihovny.



Obrázek 5.1: Struktura komponent

5.2.2.1 Reprezentace stavu

Stav se ukládá do stromové struktury, kdy každý modul (=uzel stromu) má vlastní část stavu, gettery, mutace, akce a moduly.

Stav modulu je JS objekt, definuje se výchozí stav, který lze měnit pouze mutacemi nebo akcemi.

Gettery mají jako vstup stav modulu ze kterého vrací požadovaná data.

Mutace je metoda modulu měnící stav modulu a musí mít unikátní název v rámci celého stromu. Lze do ní předat vstupní data a nelze z ní vyvolat jiné mutace ani akce. Z pravidla má být synchronní (žádné dotazy na api apod.).

Akce představuje větší změnu stavu. Jako mutace má unikátní název napříč stromem stavu. Je robustnější než mutace, v tom smyslu, že z ní lze měnit stav, volat mutace nebo jiné akce a to i z jiného modulu.

Jako poslední lze definovat moduly, což jsou v podstatě jen potomci daného modulu ve stavovém stromě.

Na ukázce kódu 3 je načtení *vuex* do *vue* a vytvoření stavového stromu s jedním modulem.

5.2.2.2 Čtení a modifikování stavu z komponent

Po načtení *vuex* do *vue* je v komponentách k dispozici store objekt jako `this.$store`. Stav aplikace je k dispozici přes `this.$store.state` kde je kořenový modul. Vyvolat mutaci lze pomocí `this.$store.commit('nazevMutace',`

```
import Vue from 'vue';
import Vuex from 'vuex';

Vue.use(Vuex);

const store = new Vuex.Store({
    state: {
        count: 0
    },
    mutations: {
        increment(state) {
            state.count++;
        }
    }
});
```

Zdrojový kód 3: Ukázka inicializace stavu pomocí Vuex

`nepovinnyParametr)` a akci pomocí `this.$store.dispatch('nazevAkce', nepovinnyParametr)`. U vyvolání mutace nebo akce se neřeší v jakém je modulu, protože musí mít unikátní název.

Vuex také poskytuje pomocné funkce, kterými lze rychle mapovat stav, gettery, mutace i akce (resp. mapState, mapGetters, mapMutations, mapActions). Hodí se především pokud pracujeme s částí stavu více nebo chceme vyvolat mutaci přímo ze šablony, protože přes mapování přiřadíme hodnotu stavu/getteru nebo funkci mutace/akce přímo do kontextu komponenty.

5.2.2.3 Použití v aplikaci

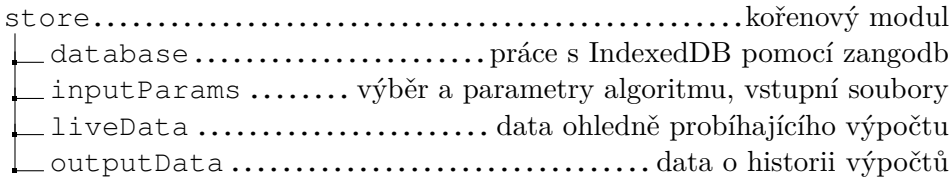
Rozdělení stavu do modulů je znázorněno na obrázku 5.2. V kořenovém modulu je uložen výsledek ověření potřebných rozhraní prohlížeče pro běh aplikace. Výsledek lze v aplikaci zobrazit.

Jeden modul je použit pro ukládání dat do databáze prohlížeče. Podobný přístup jsem jinde neviděl, ale dává mi logicky smysl. Jelikož vuex slouží pro uložení stavu, databáze dělá to samé, akorát ještě k tomu perzistentně. Modul tedy místo objektu se stavem komunikuje s databází a zbytek logiky je téměř stejný. Pro práci s databází jsme použili knihovnu *zangodb*. Do databáze ukládáme vstupní instance a výsledky běhu (a průběhu) algoritmů.

Ostatní moduly názvem víceméně hovoří samy za sebe.

5.2.3 Routing

Vzhledem k interaktivitě a provázanosti je celá aplikace jako domovská stránka (landing page). Jedinou samostatnou stránku tvoří nápověda. Ta nemá za



Obrázek 5.2: Stromová struktura stavu

účel kompenzovat nedostatky aplikace, ale poskytnout vysvětlení vybraných detailů algoritmu případným zájemcům.

```
import Vue from 'vue';
import Router from 'vue-router';
import Main from '@/components/Main';
import Help from '@/components/_help/Help';

Vue.use(Router);

export default new Router({
    routes: [{
            path: '/',
            name: 'Main',
            component: Main
        }, {
            path: '/help',
            name: 'Help',
            component: Help
    }]
});
```

Zdrojový kód 4: Router aplikace

Router aplikace (ukázka kódu 4) tedy obsahuje pouze dvě cesty: `main` a `help`. V rámci `main` hraje klíčovou roli stav, ve kterém je mimo jiné také výběr algoritmu a problému. Aby uživatel o stav nepřišel (například při obnovení stránky), ukládají se nutné údaje do local storage. Uživatel tak může v klidu přepínat mezi algoritmy nebo obnovit stránku bez toho, aby přišel o stav aplikace.

5.2.4 Web Workers

Tuto technologii integrujeme do aplikace pomocí modulu *worker loader* pro *Webpack* a funguje následujícím způsobem. Při potřebě náročnějšího výpočtu se vytvoří instance třídy `Worker`, která má jako parametr cestu k souboru s kódem (JS) pro výpočetní vlákno. Mezi hlavním a výpočetním vláknem lze

Tabulka 5.1: Seznam problémů

Problém	Cíl	Typ konfigurace	Aprox. třída
Splnitelnost boolovské formule	Max	Bitové pole	APX-úplný
Obchodní cestující	Min	Permutace	NPO-úplný
Problém batohu	Max	Bitové pole	FPTAS
Minimální uzlové pokrytí grafu	Min	Bitové pole	APX-úplný
Euklidovský obchodní cestující	Min	Permutace	PTAS

komunikovat pomocí posílání a přijímání zpráv. Dále lze z hlavního vlákna libovolné výpočetní vlákno kdykoliv ukončit.

Pro správu stavu výpočtu a pro obohacení komunikace pomocí identifikace typu zprávy jsem vytvořil vlastní rozhraní. Toto rozhraní jen obaluje a ve výsledku zjednodušuje práci s výpočetními vlákny a s řešením komunikace na obou stranách.

5.3 Metody a problémy

Veškerá logika související s problémy k řešení (např. SAT, TSP) a výpočetními algoritmy je v projektu, v adresáři `computation`. Jinými slovy je kompletně oddělena od logiky komponent a jakýkoliv výpočet se děje v rámci samostatného vlákna.

Seznam algoritmů a seznam problémů je definován v jednom souboru, který lze importovat kamkoliv, kde tyto informace potřebujeme. Slouží také jako číselník (enum), aby nebyly hodnoty v kódu napevno „zadrátované“ (hard-coded). Vytvořil jsem také pomocné metody sloužící například k získání všech informací o problému na základě identifikátoru problému. Obdobná pomocná metoda je k dispozici také pro vybraný algoritmus.

Problémy implementované v aplikaci a jejich charakteristické vlastnosti (typ problému a typ konfigurace) jsou v tabulce 5.1.

5.3.1 Vstupní instance

Uživatel má vždy k dispozici seznam instancí aktuálně vybraného problému. Pokud načte stránku a nemá pro daný problém žádné instance, aplikace do seznamu přidá instanci ukázkovou.

Pro přidání nové instance je k dispozici tlačítko pro nahrání souboru a podpora „táhni a pusť“ (drag&drop). V obou případech lze přidat více souborů najednou. Po potvrzení výběru souborů se soubory přečtou, zkontroluje se jejich formát a uloží se do databáze (zangodb). Pokud je formát validní, jsou

k dispozici základní informace o instanci (informace jednoduše zjistitelné po přečtení souboru instance). Pokud formát validní není, soubor je podbarven červeně a je k dispozici orientační zpráva, kde je pravděpodobně chyba.

Další možností je vygenerování instance. Vedle tlačítka pro výběr souborů je tlačítko pro vygenerování nové instance. Po kliknutí se zobrazí modal okno s generátorem. Generátor obsahuje vždy výběr problému a parametry pro generování vybraného problému. Následuje tlačítko „generate“.

Protože chceme, aby byla možnost generovat jednu nebo více instancí, dáváme uživatelům na výběr. Buď jen zmáčknu tlačítko „generate“, proběhne generování instance a následně se okno generátoru zavře nebo zaškrtnou checkbox „Keep generator window open“ a po vygenerování instance zůstane generátor otevřený. Generátor lze také kdykoliv zavřít bez ztráty vyplněných parametrů.

V obou případech se vygenerovaná instance přidá do seznamu a zobrazí se zpráva o úspěšném vygenerování. Jelikož lze generovat i větší instance, generátor běží na vlastním vlákně a lze ho kdykoliv v průběhu zastavit. Princip je stejný jako při spuštění výpočtu algoritmu, který popíšu v následující sekci.

Všechny nahrané i vygenerované soubory jsou zobrazeny v seznamu v pořadí nahrání. Každý soubor má u sebe ikonku pro stažení a ikonku pro odstranění.

Instance se vybírá kliknutím na název (resp. na řádek mimo zmíněné ikonky), který se barevně zvýrazní. Pokud je vybraná instance nevalidní, zobrazí se pouze zpráva s informací o nevalidním formátu.

Odstranění jednoho souboru se provede okamžitě s možností vrátit poslední akci zpět. K dispozici je také tlačítko pro odstranění všech instancí k problému. Tato akce však vyžaduje potvrzení pomocí modal okna.

5.3.2 Proces při realizaci výpočtu

Tento proces je zahájen kliknutím na tlačítko *start*. Na tlačítko *start* nejde kliknout, pokud jsou špatně zadané parametry, takže ověřování vstupu na straně výpočetního vlákna bude velmi zjednodušené.

5.3.2.1 Inicializace vlákna pro výpočet

Stisknutí tlačítka vyvolá metodu, která nejprve shromáždí všechna vstupní data. Vstupní data zahrnují vybraný algoritmus, jeho parametry, vybraný problém a obsah vybrané instance (vstupní soubor). Pokud jsou tato data validní, inicializuje se výpočetní vlákno (instance třídy *Worker*) s kódem zprostředkovávajícím výpočet zvolené instance problému vybraným algoritmem (`IterativeMethodWorker.js`). Zároveň jsou již předdefinované metody pro přijetí zpráv od výpočetního vlákna díky implementaci vlastního rozšířeného rozhraní. Vytvořené instanci se následně pošle zpráva s identifikátorem označujícím zadání práce a se všemi vstupními daty.

Důležité je, že v jakékoliv fázi běhu algoritmu ho lze ukončit tlačítkem *cancel*, které je k dispozici místo tlačítka *start* až do přijetí výsledku od výpočetního vlákna. Přijetí výsledku zároveň značí konec běhu algoritmu a hlavní vlákno pošle výpočetnímu signál k ukončení. Ukončení může ještě způsobit chyba v rámci výpočetního vlákna (např. chyba při syntaktické analýze instance).

Na straně výpočetního vlákna je poslaný identifikátor rozpoznán a podle něj se zavolá metoda odpovídající zadání práce. Ta obstará rozpoznání vybraného problému a algoritmu.

Výpočetní vlákno posílá několik typů zpráv, kterými dává hlavnímu vláknu vědět, co se děje. Tyto typy jsou: chyba, zahájení běhu algoritmu, průběh výpočtu a výsledek výpočtu.

Po rozpoznání vybraného problému se vytvoří instance, které se do konstruktoru předá obsah vstupního souboru. V konstruktoru proběhne syntaktická analýza vstupu.

Po rozpoznání vybraného algoritmu se vytvoří jeho instance, které se předají jeho vstupní parametry. Na této instanci se zavolá metoda `solve`, které se předá instanci problému.

5.3.2.2 Jednotné rozhraní problémů

Aby vše fungovalo, všechny problémy musí implementovat jednotné rozhraní. Pokročilé iterativní algoritmy pracují se všemi problémy stejně. Volají metody instance problému, které se starají o implementaci problémové části logiky.

Problém poskytuje metodu `getConfiguration`, díky které algoritmus získá instanci konfigurace. Konfiguraci lze získat náhodnou nebo výchozí („prázdnou“ - záleží na kontextu problému). Pro procházení stavového prostoru se volají metody konfigurace, které vracejí konfiguraci odvozenou. Opět platí, že každý typ konfigurace má svou specifickou implementaci.

Pro získání ohodnocení konfigurace jsou u problému k dispozici dvě metody. Jedna vrací hodnotu pro výpočet, tu používají interně všechny algoritmy. Druhá vrací reálnou hodnotu cenové funkce, která je zobrazena v grafu. Důvod tohoto rozdělení je prostý, nepotřebujeme rozlišovat zda se jedná o maximalizační nebo minimalizační problém. Jednoduše jsme implementovali algoritmy pro maximalizaci a pokud se jedná o minimalizační problém, metoda vracející hodnotu pro výpočet převrátí hodnotu reálné cenové funkce.

Pomocí těchto metod lze implementovat logiku všech tří vybraných pokročilých iterativních algoritmů a daly by se implementovat i další. Stejně tak lze v budoucnu přidat i další problémy.

5.3.2.3 Abstrakntní pohled na průběh algoritmu a jeko výsledek

Vrátím se k průběhu procesu výpočtu. Metoda `solve` obsahuje logiku výpočtu, při kterém se průběžně odesílají data do hlavního vlákna (UI). Děje se

tak pomocí zpráv s identifikátorem pro data průběhu algoritmu. Po doběhnutí samotného výpočtu vrátí metoda nejlepší nalezené řešení, které výpočetní vlákno pošle do hlavního vlákna. Použije při tom identifikátor pro výsledek. Tím hlavní vlákno ví, že algoritmus úspěšně doběhl a ukončí vlákno výpočetní.

Hlavní vlákno po dobu běhu algoritmu přijímá zprávy o průběhu, které přidává do stavu modulu `liveData`. Tím se také automaticky zobrazují do grafu průběhu. Kromě toho uživatel nemůže měnit parametry, ani procházet historii nebo jiné algoritmy. Může jen předčasně zastavit výpočet.

Při přijetí zprávy s výsledkem má hlavní vlákno již všechny potřebné informace o výpočtu k dispozici. Vyvolá tedy akci uložení výpočtu do historie, tím se dostanou data do databáze a vymaže se stav modulu `liveData` pro případný další běh. Také se rovnou vybere aktuálně dokončený výpočet v historii a tím se celý běh algoritmu zobrazí, což je v podstatě stejný graf jako na konci průběhu, ale s možností přidávat jiné běhy z historie do porovnání. Dále je k dispozici zobrazení nejlepší konfigurace, její hodnota, čas běhu výpočtu a parametry, se kterými byl algoritmus spuštěn.

5.4 Optimalizace operací souvisejících s během algoritmu

Kromě samotného běhu algoritmu aplikace poskytuje uživateli data o průběhu vybraného algoritmu. S tím souvisí hlavně komunikace mezi výpočetním vláknem a hlavním vláknem pro UI a vykreslování grafu. V důsledku bylo zapotřebí přizpůsobit například formát ukládání dat a práci s nimi v rámci výběru porovnání apod. Nicméně hlavní efekt spočíval v již zmíněných procesech: upravení posílání zpráv o průběhu algoritmu a v práce s vykreslováním dat do grafu.

Optimalizaci těchto dvou procesů jsem se věnoval značnou dobu. Abych se k optimalizaci samotné vůbec dostal, musel jsem zjistit, že právě tyto procesy jsou klíčové pro výkon aplikace. K tomu jsem použil ladící nástroje v prohlížeči (Chrome), ale nejvíce šlo o testování s postupnými úpravami kódu.

5.4.1 Vykreslování grafu

Vyzkoušel jsem několik knihoven pro vizualizaci dat. Důležitým aspektem byla možnost dynamicky měnit data a také efektivita změny dat. Například některé knihovny řeší změnu dat překreslením celého grafu.

5.4.1.1 Porovnání typů vizualizačních knihoven

Dynamická vizualizace dat se ve webovém prostředí řeší prakticky dvěma způsoby [17] [18].

Jednou možností je použití HTML5 elementu `canvas`. Ten poskytuje JavaScriptu API pro vykreslování grafiky (2D i 3D). Pracuje se s ním jako s gra-

fickým nástrojem. Výhodou je možnost vykreslování libovolných objektů či tvarů, vývojář si ale musí sám řešit překreslování a mazání objektů. Jakmile se na plátno příkazem vykreslí objekt, na plátně je reprezentován pixely. Nelze s ním „pohnout“ jinak než smazat pixely na původní pozici a vykreslit pixely pro pozici novou nebo smazat obsah plátna a vykreslit vše znovu.

Druhou možností je použití formátu SVG [19]. Jedná se o formát 2D vektorové grafiky, která je zapsána pomocí XML. Při použití pro vykreslení dat dynamicky se tento formát nepoužije v HTML elementu `img`, ale inline, tedy přímo v HTML. S elementy SVG lze následně pracovat podobně jako s elementy HTML. Výhoda spočívá v práci s elementy SVG stejně jako libovolný JS framework pracuje s elementy HTML. Jde o práci se standardním API prohlížečů, které je v každém prohlížeči více či méně, ale přesto dobře optimalizované. O vykreslování se stará samotný prohlížeč, u kterého lze předpokládat dobrou optimalizaci. Navíc se může jednat také o optimalizaci na architektonicky nižší úrovní než je JavaScript. Nevýhodou je omezení na 2D vektorovou grafiku, což je ale pro náš účel vyhovující.

Canvas má oproti *SVG* ještě jednu výhodu a to při vykreslování velmi velkého počtu objektů do malého prostoru. Prohlížeč tak zobrazí jen výsledné pixely a neřeší celý výsledek překrývání třeba několika tisíc objektů tvořených elementy SVG. Na druhou stranu SVG má výhodu v animování objektů, canvas je nutné vždy překreslit. Takové funkce knihovny pro vizualizaci dat (pomocí canvas) zatím nemají. Interaktivita SVG je mnohem jednodušší a toho spousta knihoven používající tuto metodu využívá.

Výsledkem mého testování knihoven na vizualizaci dat vzešla knihovna *dc.js* využívající SVG. Ta je použita na oba typy grafu (real-time průběh algoritmu a porovnání průběhů).

U real-time vizualizace lze efektivně přidat malé množství objektů reprezentující nová data z průběhu. Překreslení grafu obsahuje pouze část křivky a je tak výkonově efektivní. Díky tomu lze plynule vykreslovat graf až s 20 tisíci body na ose x na běžném školním počítači.

Pro porovnání průběhů spočívá výhoda ve zvýraznění dat patřícím k běhu po najetí na legendu. Tato funkce je pro knihovny s vykreslováním pomocí element canvas náročnější. Druhou výhodou je možnost dodatečných vlastních úprav, jelikož SVG elementy jsou přístupné kdekoliv (pomocí JS) v rámci zobrazované stránky jakožto součást DOM (Document Object Model).

5.4.1.2 Práce s daty a vykreslování grafu

Dalším důvodem pro výběr knihovny *dc.js* byla optimalizace přidávání a odebírání hodnot z množiny dat (a následně grafu). Pro tento účel je v *dc.js* interně použita knihovna *crossfilter* a definují se pomoci ní dimenze a grupy nad daty.

Výňatek z kódu komponenty pro demonstraci inicializace grafu, kterou následně popisuji je na ukázce kódu 5.

Nejdříve jsem vytvořil instanci grafu pomocí volání metody knihovny pro příslušný typ grafu, té jsem předal selektor elementu, ve kterém bude graf umístěn. Tato metoda vrací objekt definující graf, na který se později aplikují metody upravující vlastnosti grafu.

Veškerá data pro graf se přidávají do crossfilteru jako pole obsahující objekty. Poté se definuje dimenze, které se jako parametr předá funkce transformující objekt z pole dat na klíč (číslo nebo řetězec znaků). Tento klíč slouží k identifikaci hodnotu na ose x. Na dimenzi se definuje grupa, které se také jako parametr předá funkce. Ta transformuje objekt na hodnotu pro podmnožinu dat patřící k jednotlivým běhům.

Po definici dimenze a grupy můžeme tyto údaje spolu s několika dalšími povinnými přidat do instance grafu. Na modifikaci nejrůznějších atributů grafu jsou k dispozici metody sloužící jako settery. Jakmile máme vše nastaveno, můžeme graf vykreslit pomocí metody `render`.

```
import dc from 'dc';

this.liveLineChart = dc.lineChart('#liveChart');

this.ndx = dc.crossfilter([
    {index: 1, value: 2},
    {index: 5, value: 3},
    ...
]);
let dim = this.ndx.dimension(d => d.index);
let grp = dim.group().reduceSum(d => d.value);

this.liveLineChart
    .width(500).height(300)
    .x(d3.scale.linear().domain([0, 100]))
    .dimension(dim)
    .group(grp);

this.liveLineChart.render();
```

Zdrojový kód 5: Inicializace dc grafu (výňatek z vue komponenty)

Když máme referenci na crossfilter použitý v grafu, lze do něj přidávat data pomocí metody `add` nebo odstranit pomocí `remove` (ukázka 6). Samotná úprava dat ještě není aplikována na graf. Pro zobrazení změn je nutné zavolat na instanci grafu metodu `render` nebo `redraw`.

Mezi metodami `render` a `redraw` je zásadní rozdíl. Metoda `render` vykreslí celý graf od začátku, proto ji také musíme použít pro první vykreslení grafu. Metoda `redraw` hraje klíčovou roli pro výkon aplikace. Tato metoda totiž umí zjistit jaké změny dat byly pomocí crossfilteru provedeny a na základě této informace přidat nebo odebrat část grafu tak, aby odpovídala ak-

tuálním údajům. Nedochází zbytečně k odstraňování a přidávání elementů, pouze se změní jejich vlastnosti. Tím lze dosáhnout poměrně efektivní změny v grafu.

```
this.ndx.add([
    {index: 10, value: 5},
    {index: 25, value: 2},
    ...
]);

this.liveLineChart.redraw();
```

Zdrojový kód 6: Přidání dat do grafu pomocí crossfilteru (použití ve vue komponentě)

Dalším testováním jsem zjistil, že vykreslování každého bodu tak, jak po každé iteraci přicházejí od algoritmu nepřináší optimální výkon. Zkusil jsem hodnoty do grafu přidávat jednou za 50 nebo 100 milisekund. Tím jsem se z plynulého vykreslování 10 tisíc bodů dostal k 20 tisícům bodů. Nakonec jsem se dostal k podobnému problému i u posílání zpráv od výpočetního vlákna. Tento problém a jeho řešení blíže popíšu v následující sekci. Řešením na straně hlavního vlákna bylo přidávání nezpracovaných hodnot do pomocného pole. Pokud uběhla minimální doba od posledního vykreslení grafu, přidat hodnoty, vykreslit graf a aktualizovat poslední dobu vykreslení. Jelikož se kontrola posledního vykreslení provádí pouze při změně dat, je potřeba ještě ošetřit konec algoritmu. Při přijetí výsledku se poslední data o průběhu a výsledek uloží a vykreslí se celý graf připravený pro porovnávání.

Nakonec optimalizace grafu bych ještě dodal, že proces vykreslování nelze provádět v jiném vlákně. Jiné než hlavní vlákno (tj. `Worker`) nemá přístup k zobrazovaným elementům. Proto je nutné optimalizovat vykreslování, aby nedošlo k prodlevách vykreslení, kvůli kterým by uživatel dočasně ztratil kontrolu nad UI.

5.4.2 Posílání zpráv o průběhu algoritmu

V prvotní implementaci posílal algoritmus běžící v rámci výpočetního vlákna zprávu o průběhu každou iteraci. Důvodem byla intuitivní a jednoduchá (naivní) implementace. Testováním jsem ale zjistil, že komunikace mezi vlákny není v rámci výkonu „zadarmo“ a reálně ani není nutná takto častá komunikace. Řešení je podobné jako v případě vykreslování grafu, jen v jiném kontextu.

Implementoval jsem třídu pro shromažďování zpráv. Na začátku se vytvoří instance, které se v konstruktoru předá rozhraní pro komunikaci a minimální čas mezi zprávami. Následně se místo přímého posílání volá metoda, pomocí

které se přidávají zprávy k odeslání a pokud uběhne zadaný čas, odešlou se nasbírané hodnoty.

Čas mezi odesláním zpráv jsem po testování zvolil 75 milisekund. Větší čas by mohl způsobit méně plynulé vykreslování a menší čas zbytečnou komunikaci. Při zvolení delšího intervalu se logicky nasbírá více hodnot a překreslení grafu tak obsahuje větší změnu pro zpracování. Proto byl zvolen poměrně malý interval, který však nakonec výkon výrazně zvýšil.

Pomocí optimalizace na obou stranách bylo dosaženo plynulého vykreslování až okolo 25 tisíc bodů. Pro porovnání: knihovna *chart.js* používající canvas a bez použití podobné optimalizace jako nabízí crossfitler měla značný problém už při vykreslování množiny dat o mohutnosti menší než tisíc bodů.

5.5 Problém splnitelnosti booleovských formulí

Již jsem popisoval jednotné rozhraní problémů, nyní se zaměřím na problém SAT, který jsem pro aplikaci implementoval.

5.5.1 Definice

Jedná se o optimalizační SAT pro formule v konjunktivní normální formě.

- Je dána booleovská formule F proměnných $X = (x_1, x_2, ..., x_n)$ v konjunktivní normální formě (tj. součin součtů).
- Nalezněte ohodnocení $Y = (y_1, y_2, ..., y_n)$ proměnných $x_1, x_2, ..., x_n$ tak, aby počet splněných klauzulí formule F byl maximální.

5.5.2 Implementace

Vstup je ve formátu DIMACS kvůli strojové čitelnosti a kompatibilitě s instancemi používanými v rámci předmětu MI-PAA. Podrobnosti o formátu a spousta instancí je k dispozici na adrese:

http://www.cs.ubc.ca/~hoos/SATLIB/benchm.html

Bez ohledu na přesný formát musí být ze vstupu jasné, kolik má formule proměnných a kolik má klauzulí. Kvůli srozumitelné logice jsem vytvořil třídy reprezentující *literál* a *klauzuli* (ukázka kódu 7). Instance problému je tak reprezentována polem klauzulí (obsahující literály).

Cenová funkce (počítá metoda `evaluateMaximizationCost`) odpovídá počtu splněných klauzulí (ukázka kódu 8). Řešení instance problému má tedy cenovou funkci rovnou počtu svých klauzulí. Metoda pro výpočet cenové funkce vyžaduje jako parametr konfiguraci problému. Konfigurace je v podstatě bitové pole odpovídající ohodnocení proměnných. Samotný výpočet ceny spočívá v iteraci přes instance klauzulí. Na každé klauzuli se volá metoda

```
class Literal {
    constructor(id, inv) {
        this.id = id;    // int
        this.inv = inv;  // boolean "inverted"
    }
}

class Clause {
    constructor(literalArray) {
        this.literals = literalArray.map(literal => {
            let inv = (literal < 0);
            let id = inv ? -literal : +literal;
            return new Literal(id - 1, inv);
        });
    }

    check(bitArray) {
        for (let literal of this.literals) {
            if (bitArray[literal.id] && !literal.inv) return true;
            if (!bitArray[literal.id] && literal.inv) return true;
        }
        return false;
    }
}
```

Zdrojový kód 7: Pomocné třídy pro reprezentaci instance problému SAT

`check`, které se předá ohodnocení proměnných. Tato metoda zkontroluje každý literál a zjistí, zda je alespoň jeden z nich vyhodnocen jako pravdivý, což odpovídá splnění klauzule. Stačí tedy sečíst počet splněných klauzulí a vrátit výsledek.

```
import { BitArray } from "./configurationTypes/BitArray";
import { Problem, ProblemTypeEnum } from './Problem';

export class SAT extends Problem {
    constructor(data) {
        super();
        this._clauses = [];
        let dataSet = data.split(/\s*(0|\n)\s*/)
            .filter(row => row.trim()[0] !== 'c');
        let params = dataSet[0].split(/(\s+)/)
            .filter(x => x.trim().length > 0)
            .splice(2, 2).map(param => +param);

        this.params = {
            numberOfVariables: params[0],
            numberOfClauses: params[1]
        };

        dataSet = dataSet.splice(1, params[1]).map(row =>
            row.trim().split(' ')
                .splice(0, row.length - 1)
                .map(number => +number)
        );

        for (let row of dataSet) {
            this._clauses.push(new Clause(row));
        };
    }

    _check(bitArray) {
        return this._clauses.reduce((sum, clause) => {
            return(clause.check(bitArray)) ? sum + 1 : sum
        }, 0);
    }

    evaluateMaximizationCost(bitArrayConfig) {
        const bitArray = bitArrayConfig.getBitArray();
        return this._check(bitArray);
    }

    transformMaximizationToRealCost(maxCost) {
        return maxCost; // already a maximization problem
    }

    getConfiguration(random) {
        return new BitArray({ size: this.params.numberOfVariables,
            random: random });
    }
    // ... rest of the problem interface
    // (mostly helper methods and getters)
}
```

Zdrojový kód 8: Zkrácená verze třídy reprezentující instanci problému SAT

Kapitola 6

Uživatelské testování

Testování prototypu bylo provedeno v rámci návrhu uživatelského rozhraní. Po implementaci je však na místě otestovat aplikaci se vším všudy. Testování bylo rozděleno na čtyři části. Nejprve jsme otestovali UI včetně doplněné a rozšířené funkčnosti (oproti prototypu), na kterou jsme se zaměřili. Následovalo testování jednotlivých algoritmů jakožto sekcí aplikace zvlášť.

6.1 Příprava testování

6.1.1 Společná část

Pro společnou část jsme rozšířili otázky z testování prototypu a přibyly hlavně otázky k jednotlivým algoritmům. Testeři mohli také hodnotit celkový dojem z aplikace včetně funkčnosti.

Při testování jsme si rozdělili role. Michal se staral o procházení scénářů a celkově komunikaci s testery [1]. Zapisoval také připravené otázky. Adam zapisoval všechny informace, které nám tester v průběhu testování sdělil verbálně [2]. Já jsem se staral především o neverbální projev testera. Zapisoval jsem kdy a v jaké souvislosti se v aplikaci ztrácel. Jaké věci byly naopak zajímavé a intuitivní.

Testování se zúčastnilo 7 studentů magisterského studia fakulty informačních technologií.

Každému testerovi jsem na začátku testování vysvětlil/připomněl, že netestujeme jejich schopnosti, ale aplikaci.

6.1.2 Tabu prohledávání

Sekci Tabu prohledávání jsem chtěl pojmout jinak. Nechtěl jsem postupovat scénářem s detailně popsanými kroky. Využil jsem situace, kdy testeři s aplikací pracovali přibližně 15 minut. Chtěl jsem zjistit jak hladce proběhne pravděpodobně běžný proces použití.

Scénář jsem schválně navrhl velmi stručný, aby popisoval jen to, čeho budou uživatelé běžně chtít docílit. Žádné nadbytečné informace a hlavně žádná informace, která by uživatele nějakým způsobem vedla k dosažení výsledku.

Scénář byl následující:

1. Vyberte si libovolný problém, na kterém budete zkoumat Tabu prohledávání.
2. Pomocí generátoru si připravte instanci problému.
3. Spusťte třikrát běh jedné instance pokaždé s různými parametry.
4. Porovnejte výsledky běhů.

Na základě tohoto jednoduchého scénáře jsem pozoroval, jak dobře se studentům s aplikací pracovalo po poměrně krátké době. Především jsem se zaměřil na orientaci. Zda testovaný uživatel znovu hledá, kde se daný ovládací prvek nachází nebo již přesně ví kam kliknout.

Kromě poznatků z chování jsem testování zakončil následujícími otázkami:

1. Jak hodnotíte náročnost práce s aplikací?
2. Myslíte si, že by Vám aplikace pomohla k rozšíření znalostí Tabu prohledávání?
3. Chybí Vám v aplikaci nějaké informace?

6.2 Otázky před a po testování

Všichni testeři absolvovali předmět MI-PAA.

Otázky 1–5 byly kladeny před testováním a otázky 6–12 po testování.

Eva

1. Jaký algoritmus jste si vybrali pro svou závěrečnou práci v předmětu MI-PAA?
   - Simulované ochlazování.
2. Znáte i zbylé dva algoritmy?
   - Pouze jsem o nich slyšela.
3. Znáte bývalé applety, které sloužily k demonstraci běhu iterativních algoritmů?
   - Neznám.
4. Zúčastnili jste se testování prototypu naší aplikace?

- Ano.

5. Prohlédněte si aplikaci a sdělte nám, jak byste aplikaci začali používat.
   - Začala bych proklikáním záložek a poté bych spustila běh jednoho z nich.
6. Co se vám na aplikaci líbilo?
   - Pěkné UI, grafová reprezentace.
7. Co se vám nelíbilo, nebo co vám přišlo složité / neintuitivní?
   - Instance se špatným formátem je těžké poznat.
8. Co si o této aplikaci myslíte v porovnání s původními applety?
   - Původní applety jsem neviděla.
9. Myslíte si, že by vám tato aplikace pomohla pochopit jednotlivé algoritmy a vliv parametrů na jejich běh?
   - Spíše ano.
10. Jak hodnotíte náročnost práce s aplikací?
    - Přiměřená.
11. Myslíte si, že by Vám aplikace pomohla k rozšíření znalostí Tabu prohledávání?
    - Spíše ano.
12. Chybí Vám v aplikaci nějaké informace?
    - Asi ne.

Marek

1. Jaký algoritmus jste si vybrali pro svou závěrečnou práci v předmětu MI-PAA?
   - Simulované ochlazování.
2. Znáte i zbylé dva algoritmy?
   - Znám.
3. Znáte bývalé applety, které sloužily k demonstraci běhu iterativních algoritmů?

- Znám.

4. Zúčastnili jste se testování prototypu naší aplikace?
   - Ano.
5. Prohlédněte si aplikaci a sdělte nám, jak byste aplikaci začali používat.
   - Spustil Start.
6. Co se vám na aplikaci líbilo?
   - Je barevná.
   - Dobrá vizualizace.
   - Nezasekává se.
7. Co se vám nelíbilo, nebo co vám přišlo složité / neintuitivní?
   - Instance se špatným formátem je těžké poznat.
   - Čísla u stejně pojmenovaných instancí v historii.
8. Co si o této aplikaci myslíte v porovnání s původními applety?
   - Lepší, snadno spustitelné.
9. Myslíte si, že by vám tato aplikace pomohla pochopit jednotlivé algoritmy a vliv parametrů na jejich běh?
   - Spíše ano.
10. Jak hodnotíte náročnost práce s aplikací?
    - Jednoduchá.
11. Myslíte si, že by Vám aplikace pomohla k rozšíření znalostí Tabu prohledávání?
    - Pravděpodobně ano.
12. Chybí Vám v aplikaci nějaké informace?
    - Nevěděl jsem co reprezentuje tabulka s T/F (tj. vizualizace nejlepšího nalezeného stavu).

Petra

1. Jaký algoritmus jste si vybrali pro svou závěrečnou práci v předmětu MI-PAA?
   - Simulované ochlazování.
2. Znáte i zbylé dva algoritmy?
   - Ano, ale už si je moc nepamatuji.
3. Znáte bývalé applety, které sloužily k demonstraci běhu iterativních algoritmů?
   - Ano.
4. Zúčastnili jste se testování prototypu naší aplikace?
   - Ne.
5. Prohlédněte si aplikaci a sdělte nám, jak byste aplikaci začali používat.
   - Prohlédla si nápovědy u jednotlivých parametrů a poté algoritmy.
6. Co se vám na aplikaci líbilo?
   - Přehledné rozhraní.
   - Spousta vysvětlivek, validace.
   - Dobré a přehledné ovládání.
7. Co se vám nelíbilo, nebo co vám přišlo složité / neintuitivní?
   - Ze začátku jsem neviděla historii, ale nemyslím si že to je problém.
8. Co si o této aplikaci myslíte v porovnání s původními applety?
   - Přehlednější, novodobé a snadno spustitelné.
9. Myslíte si, že by vám tato aplikace pomohla pochopit jednotlivé algoritmy a vliv parametrů na jejich běh?
   - Ano, dobré na učení. Člověk si to může vyzkoušet aniž by to programoval.
10. Jak hodnotíte náročnost práce s aplikací?
    - Je hezká a přehledná.
11. Myslíte si, že by Vám aplikace pomohla k rozšíření znalostí Tabu prohledávání?

- Určitě, je to zábavnější než učení pouze z přednášek.

12. Chybí Vám v aplikaci nějaké informace?
    - Na první pohled ano, ale později jsem je našla a pak už s tím problém nebyl.

Danny

1. Jaký algoritmus jste si vybrali pro svou závěrečnou práci v předmětu MI-PAA?
    - Simulované ochlazování.
2. Znáte i zbylé dva algoritmy?
    - Z toho co jsem se učila na zkoušku.
3. Znáte bývalé applety, které sloužily k demonstraci běhu iterativních algoritmů?
    - Ano, ale šly mi spatně spustit.
4. Zúčastnili jste se testování prototypu naší aplikace?
    - Ne.
5. Prohlédněte si aplikaci a sdělte nám, jak byste aplikaci začali používat.
    - Proklikala algoritmy a pak start.
6. Co se vám na aplikaci líbilo?
    - Graf, porovnání instancí.
7. Co se vám nelíbilo, nebo co vám přišlo složité / neintuitivní?
    - Historie na začátku trochu matoucí, po chvíli už bez problému.
8. Co si o této aplikaci myslíte v porovnání s původními applety?
    - Lehce spustitelné.
    - Funkčnost.
    - Lépe se s tím pracuje.
9. Myslíte si, že by vám tato aplikace pomohla pochopit jednotlivé algoritmy a vliv parametrů na jejich běh?
    - Ano, i při řešení domácích úloh.
    - Aplikace by mě více motivovala ke studiu díky zábavnější formě.

10. Jak hodnotíte náročnost práce s aplikací?
    - Normální.
11. Myslíte si, že by Vám aplikace pomohla k rozšíření znalostí Tabu prohledávání?
    - Nejspíš ano.
    - Hodí se také pro lepší výběr algoritmu pro semestrální práci.
12. Chybí Vám v aplikaci nějaké informace?
    - Nevím, asi ne.

Jan

1. Jaký algoritmus jste si vybrali pro svou závěrečnou práci v předmětu MI-PAA?
    - Simulované ochlazování.
2. Znáte i zbylé dva algoritmy?
    - Ano.
3. Znáte bývalé applety, které sloužily k demonstraci běhu iterativních algoritmů?
    - Ne.
4. Zúčastnili jste se testování prototypu naší aplikace?
    - Ne.
5. Prohlédněte si aplikaci a sdělte nám, jak byste aplikaci začali používat.
    - Zkusil bych tlačítko start.
6. Co se vám na aplikaci líbilo?
    - Rozložení.
    - Interaktivní grafy.
7. Co se vám nelíbilo, nebo co vám přišlo složité / neintuitivní?
    - Závorkování při stejných jménech.
    - Popis u najetí na graf.
8. Co si o této aplikaci myslíte v porovnání s původními applety?

- Applety jsem neviděl.

9. Myslíte si, že by vám tato aplikace pomohla pochopit jednotlivé algoritmy a vliv parametrů na jejich běh?
   - Ladit parametry pro semestrální práci.
10. Jak hodnotíte náročnost práce s aplikací?
    - Dobře.
11. Myslíte si, že by Vám aplikace pomohla k rozšíření znalostí Tabu prohledávání?
    - Myslím si, že ano.
12. Chybí Vám v aplikaci nějaké informace?
    - Popis u hodnot po najetí na graf (Tabu).

Jindřich

1. Jaký algoritmus jste si vybrali pro svou závěrečnou práci v předmětu MI-PAA?
   - Genetický algoritmus.
2. Znáte i zbylé dva algoritmy?
   - Ano, teoreticky.
3. Znáte bývalé applety, které sloužily k demonstraci běhu iterativních algoritmů?
   - Nevěděl jsem, že existují.
4. Zúčastnili jste se testování prototypu naší aplikace?
   - Ne.
5. Prohlédněte si aplikaci a sdělte nám, jak byste aplikaci začali používat.
   - Spustil bych postupně všechny algoritmy.
6. Co se vám na aplikaci líbilo?
   - Jednoduchost, přímočarost, vše na jedné obrazovce.
7. Co se vám nelíbilo, nebo co vám přišlo složité / neintuitivní?
   - Problém pod parametry (umístění v levém panelu).

8. Co si o této aplikaci myslíte v porovnání s původními applety?
   - Applety jsem neviděl.
9. Myslíte si, že by vám tato aplikace pomohla pochopit jednotlivé algoritmy a vliv parametrů na jejich běh?
   - Určitě.
10. Jak hodnotíte náročnost práce s aplikací?
    - Dobře se s ní pracuje.
11. Myslíte si, že by Vám aplikace pomohla k rozšíření znalostí Tabu prohledávání?
    - Ano.
12. Chybí Vám v aplikaci nějaké informace?
    - Popis, jaký stav je zobrazen ve spodní tabulce (tj. vizualizace nejlepšího nalezeného stavu).

Jakub

1. Jaký algoritmus jste si vybrali pro svou závěrečnou práci v předmětu MI-PAA?
   - Genetický algoritmus.
2. Znáte i zbylé dva algoritmy?
   - Ano z PAA.
3. Znáte bývalé applety, které sloužily k demonstraci běhu iterativních algoritmů?
   - Ano.
4. Zúčastnili jste se testování prototypu naší aplikace?
   - Ne.
5. Prohlédněte si aplikaci a sdělte nám, jak byste aplikaci začali používat.
   - Proklikal bych si to a spustil.
6. Co se vám na aplikaci líbilo?
   - Dynamičnost.

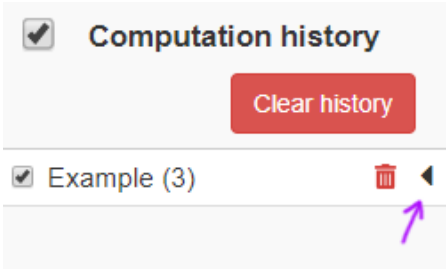
7. Co se vám nelíbilo, nebo co vám přišlo složité / neintuitivní?
   - Problém pod parametry (umístění v levém panelu).
   - Tlačítko na generátor, konkrétně ikonka.
8. Co si o této aplikaci myslíte v porovnání s původními applety?
   - Funguje.
9. Myslíte si, že by vám tato aplikace pomohla pochopit jednotlivé algoritmy a vliv parametrů na jejich běh?
   - Ano.
10. Jak hodnotíte náročnost práce s aplikací?
    - Myslím, že s tím není problém.
11. Myslíte si, že by Vám aplikace pomohla k rozšíření znalostí Tabu prohledávání?
    - Ano, je dokonce interaktivnější než bych čekal.
12. Chybí Vám v aplikaci nějaké informace?
    - Ne, ale bylo by hezké mít možnost přiblížit a oddálit graf (vybrat část grafu, na kterou se chci zaměřit).

6.3 Poznatky a vyhodnocení

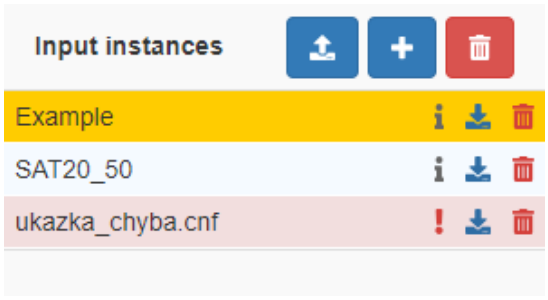
Hned první otázka - Jaký algoritmus jste si vybrali pro svou závěrečnou práci v předmětu MI-PAA? - podpořila mé tvrzení, že Tabu prohledávání není v předmětu MI-PAA pro semestrální práci příliš populární.

- Většina testerů znala z předchozího studia i zbylé algoritmy.
- Applety se většině naopak spustit nepodařilo a někteří ani nevěděli, že existují.
- Testování prototypu se účastnili 2 ze 7 testerů.

Při pohledu na aplikaci by si někteří proklikali algoritmy a někteří by si prohlédli parametry. Všichni se ale shodli, že by spustili běh nějakého algoritmu, aby viděli, co aplikace umí. To byl náš záměr, abychom studenty neodradili vyplňováním parametrů dříve, než se o algoritmy začnou zajímat, předvyplnili jsme parametry a připravili ukázkovou instanci. Student tak může rovnou spustit běh a začít experimentovat. Ověřili jsme si, že se nám tohoto efektu v rámci možností podařilo docílit.



Obrázek 6.1: Nevhodné umístění rozbalování parametrů běhu



Obrázek 6.2: Rozlišení vybrané, validní a nevalidní instance

6.3.1 Nedostatky

Nedostatky, které testeři zmínili, postupně projdu a zhodnotím podle závažnosti a častosti výskytu.

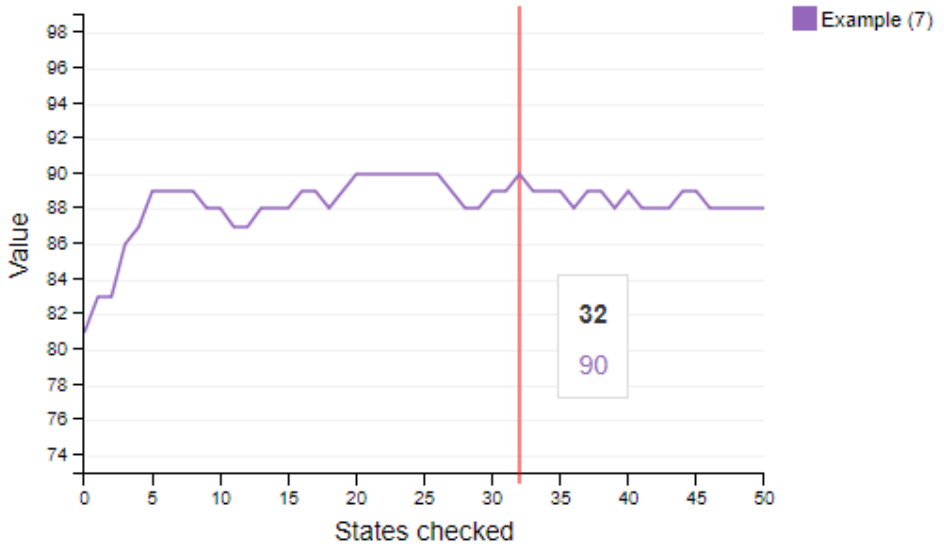
Mezi časté a závažnější nedostatky patří nepochopení tabulky výsledných řešení. Někteří vyhodnotili, že to bude asi nějaký stav, ale nevěděli jaký. Jiní si tabulku s ničím nespojili. Po konzultaci s testery jsme zjistili, že by stačilo přidat nadpis popisující význam tabulky.

Dalším častějším problémem bylo umístění šipky pro rozbalení instance (obrázek 6.1). Ikonka je umístěna vedle odstranění a uživatelům se tato volba nelíbila. Navíc je vedle barevného koše málo výrazná.

Na druhou stranu když už se některým podařilo záznam z historie odstranit, líbila se jim možnost vrátit akci zpět.

Dvěma testerům se nelíbilo vizuální zpracování nevalidní instance (obrázek 6.2). Pozadí nebylo příliš výrazné a nevšimli si vykřičníku místo ikonky info („i“). Ostatní testeři ale neměli s pochopením problém. Přes méně zřetelné podbarvení si všimli ikonky červeného vykřičníku místo šedého „i“.

S odlišením nevalidní instance jsem měl problémy a vím, že stav není ideální. U výraznější barvy zaniká text a hlavně by neměla být výraznější než vybraná instance. Volba ikonek také nebyla jednoduchá, protože není k dispozici příliš místa. Použití ikonky s obrysem je tvar deformován pixely monitoru a ikonka je špatně čitelná.



Obrázek 6.3: Interaktivní graf průběhu algoritmu (vývoj cenové funkce)

Umístění výběru problému nevyhovovalo dvěma testerům. Navrhovali umístit výběr problémů nahoru na začátek levého panelu. Oba testeři toto pozicování hodnotili jen podle intuice a řekli, že pro práci s aplikací jim to nevadí. Šlo o první dojem. V následujících krocích také pochopili, že jsme chtěli nechat pohromadě výběr problému a seznam instancí, protože se vždy zobrazují instance k danému problému.

Dalším potenciálním problémem je z počátku neočekávané umístění historie. Podobná záležitost jako s umístěním výběru problému. Znovu šlo o prvotní dojem, po chvilce už práce s historií nejevila známky problému.

Jeden tester poukázal na nekonzistenci při najetí na graf běhu Tabu prohledávání (obrázek 6.3). Oproti ostatním algoritmům chybí popis hodnoty. Popis jsem tam nedal, jelikož mi přišlo jasné, že se jedná o jedinou hodnotu v grafu. Když vezmu v úvahu konzistenci, dává mi už popis smysl a nemyslím si, že by měl naopak nějaký negativní efekt.

Nejasná ikonka generátoru. Na tuto záležitost poukázal jeden tester a v průběhu vývoje jsem na ni narazil několikrát. Vždy jsem se snažil nalézt intuitivnější ikonku a nikdy se mi to nepodařilo. Navrhoval jsem jiné ikonky běžně používané při práci se soubory. Vždy jsme se ale shodli, že mají svůj význam a v naší aplikaci by byly více matoucí, než obecná ikonka „plus“. Snížení stresu uživatelů před kliknutím jsme zajistili tooltipem po najetí na tlačítko. Tooltip nakonec zachránil situaci. Ostatní testeři generátor díky tomu rychle našli a také tester, kterému se ikonka nelíbila neměl nakonec s její funkčností další problémy.

Poslední nedostatek, který byl pojat spíše jako návrh na novou funkci je

přiblížení grafu. Tester Jakub popisoval, jak by se mu líbila možnost přiblížení části grafu, aby si mohl prohlédnout případné zajímavé úseky průběhu důkladněji.

6.3.2 Pozitiva

Upřímně jsem nečekal, že se bude testujícím studentům naše aplikace tolik líbit. Pravděpodobně mi přišla práce s aplikací po několika měsících vývoje již poměrně nudná.

Testerům se líbila jak vzhledově, tak nabízenou funkčnost. Padla hodnocení jako moderní, přehledná, přímočará a barevná. Dále testeři chválili přístupnost, interaktivitu, nápovědy na každém rohu a okamžitou validaci s jasným popisem chyby. Marek ještě pozitivně ohodnotil plynulost běhu a celkově procházení aplikací.

6.3.3 Shrnutí

Výsledky testování dopadly dle mého názoru velmi dobře. Narazili jsme na různé chyby a nedostatky, ale nic z toho nemělo za následek nesplnění primárního účelu aplikace. Většina ovládacích prvků aplikace je intuitivní a prvky, které nejsou obecně standardizované mají k dispozici nápovědu.

6.4 Konzultace aplikace s cvičícími

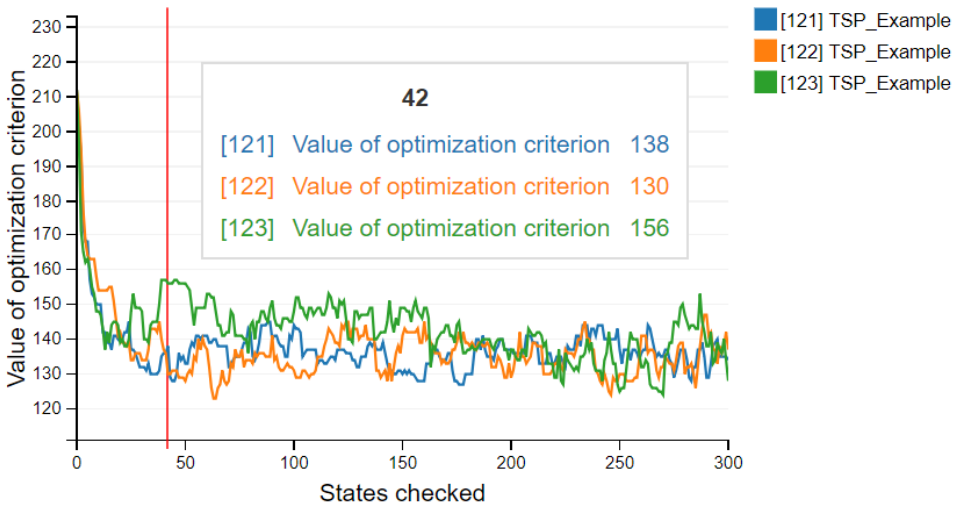
Uživateli aplikace budou primárně studenti. To ale neznamená, že ji nemohou použít cvičící k ilustraci výuky.

Cvičícím předmětu MI-PAA jsem rozeslali odkaz na nasazenou testovací verzi aplikace (stejnou jaká byla použita pro uživatelské testování studentů) s prosbou o projití aplikace a sepsání případných nedostatků nebo připomínek. Také jsme průběžně konzultovali stav aplikace s vedoucím.

Na závěrečné konzultaci byl přítomen také doc. Ing. Jan Schmidt, Ph.D., který nám pomohl sjednotit logiku názvosloví. Jeho zkušenosti nejen ohledně návrhu uživatelských rozhraní vnesly do diskuze nové poznatky. Díky nim jsme sjednotili například pojmenování běhu.

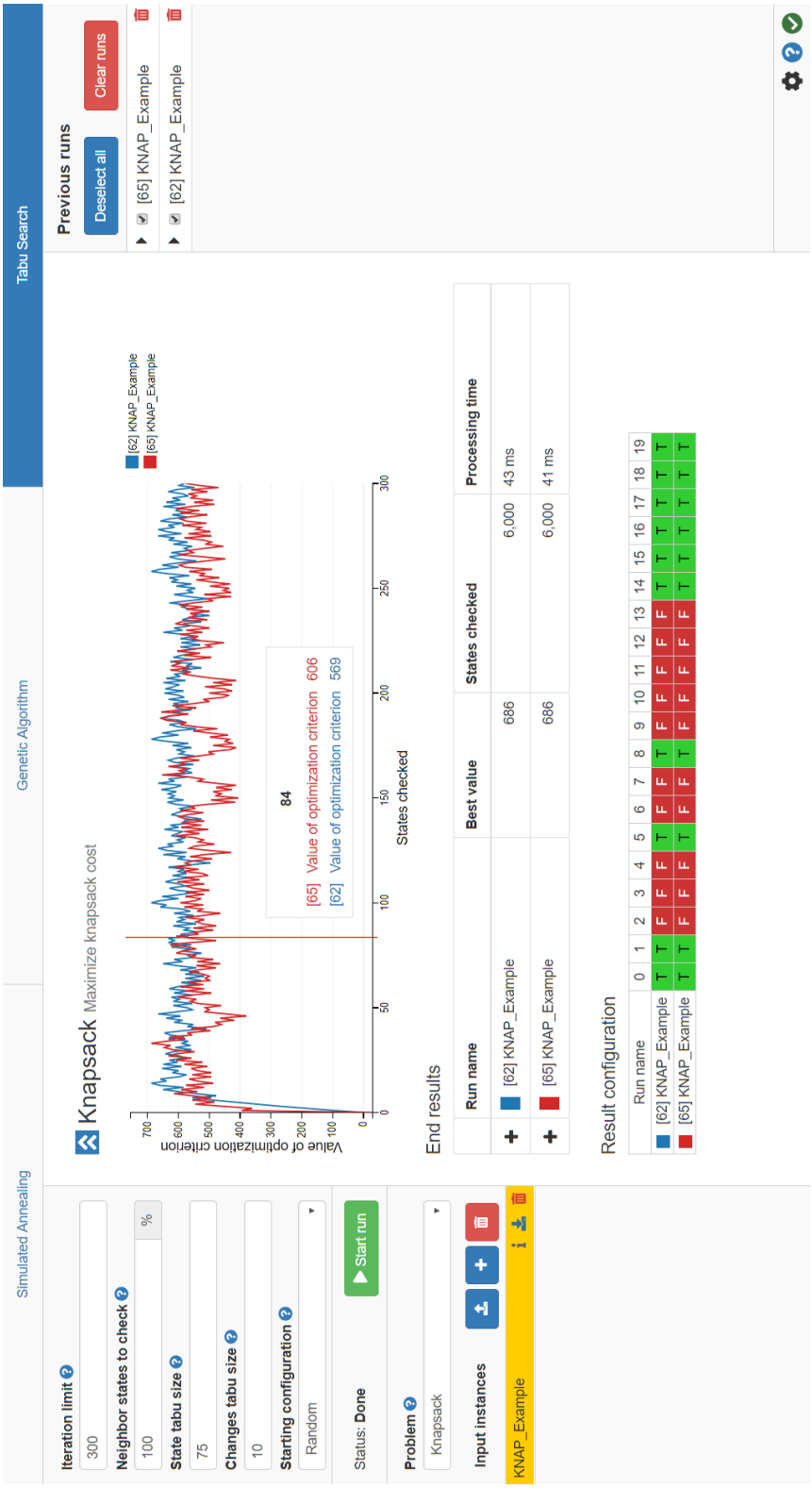
Tlačítko „start“ jsme přejmenovali na „start run“. Každý záznam běhu obsahuje kromě názvu instance také identifikátor (jeden z problémů při testování). Tím pádem jsme místo pojmenování „instance“ nebo „name“ zvolili „run name“. Problematický nadpis „computation history“ jsme přejmenovali na „Previous runs“ a k tomu korespondující tlačítko pro smazání „clear runs“. Napříč aplikací je tak zřejmá návaznost pomocí označení běhu (obrázek finální verze aplikace 6.5).

Druhý důležitý poznatek spočíval v informaci při najetí na graf. Návrh tabulky s hodnotami byl založen na párování hodnot k běhům pouze podle obarvení. To není dobré z několika důvodů, někteří lidé mají například problémy



Obrázek 6.4: Finální verze interaktivního grafu

s rozpoznáváním barev nebo stačí, když jsou nepříznivé světelné podmínky. Aplikace by tak disponovala nedostatečným označením. Pro vyřešení tohoto problému by k jednotlivým hodnotám přidán identifikátor. Rozpoznání tím pádem není závislé pouze na barevném odlišení. Výsledné zpracování je na obrázku 6.4.



Obrázek 6.5: Finální verze aplikace

Kapitola 7

Nasazení

Před nasazením aplikace je potřeba zdrojový kód ve formátu pro vývoj sestavit do uceleného, ideálně optimalizovaného a minimalizovaného kódu pro nasazení na server. Podstata sestavení aplikace spočívá nejprve v kompozici kódu aplikace s externími knihovnami a kódem frameworku. Také se spustí CSS preprocesor, v našem případě SASS, a převede formát SCSS na CSS. Následuje minimalizace kódu. Důvod je velmi prostý, každý nadbytečný znak (převážně neviditelné znaky) zabírá zbytečné místo a o to déle by trvalo načtení aplikace. Minimalizuje se jak JS tak CSS. Nakonec se vezme šablona HTML a vloží se do ní odkazy na sestavené kódy.

7.1 Webpack a jeho konfigurace pro jednotlivá prostředí

Pro proces sestavení používáme knihovnu *webpack* [20]. Logicky to vyplývá z faktu, že jsme *webpack* použili pro rozdělení kódu do modulů.

Jediným požadavkem na prostředí, ve kterém lze sestavit aplikaci, je instalace softwaru *node.js*.

Při vytvoření *Vue* projektu jsou vygenerovány dvě základní konfigurace definující sestavení. Konfigurace navíc podporují dědičnost, takže lze definovat společné nebo výchozí vlastnosti a procesy sestavení, které lze rozšířit nebo upravit pro konkrétní typ prostředí.

První konfigurace je určena pro vývojové prostředí s podporou rozšíření pro vývoj jako například hot module reload (hmr) nebo třeba mapování zdrojového kódu pro snadné hledání chyb. Hlavním cílem je interaktivní vývoj, proto nemá smysl například minimalizace kódu. Výsledkem je spuštění lokálního serveru na portu definovaném v konfiguraci.

Druhá je pro produkční prostředí. U té je kladen důraz na minimalizaci, optimální výkon a nezahrnuje jakékoliv knihovny pro usnadnění vývoje. Vý-

sledkem jsou výstupní soubory aplikace (HTML, CSS a JS) v adresáři definovaném v konfiguraci.

Pro nasazení testovací verze jsem vytvořil třetí konfiguraci. Pro fungování aplikace je potřeba znát relativní url. Ta se na produkčním serveru bude od testovacího pravděpodobně lišit, proto dává smysl vytvořit novou konfiguraci. Druhým rozdílem je cílový adresář pro výstupní soubory. Testovací verzi nasazujeme na *Github pages*, proto je nastaven výstup do adresáře `docs`.

Pro spuštění jakéhokoliv sestavení se používá následující příkaz.

```
$ npm run <název-úlohy>
```

7.1.1 Postup nasazení na github pages

Díky specifické konfiguraci stačí pro nasazení testovací verze následující postup (příkazy spustit v adresáři projektu):

- Získat aktuální kód `git pull`
- Spustit sestavení `npm run demo`
- Uložit změny `git add -all; git commit -m "Demo build"`
- Poslat změny do github repozitáře `git push`

Github rozezná úpravy souborů pro github pages a zveřejní je na adrese přiřazené k repozitáři.

7.1.2 Postup nasazení ze souborů na DVD

Pro nasazení na server musíme v konfiguraci nastavit relativní nebo absolutní cestu k umístění souborů z hlediska url. Tato cesta je nastavena pomocí proměnné `build.assetsPublicPath` v konfiguračním souboru `/src/impl/config/index.js` na DVD. Například hodnota této proměnné pro testovací prostředí je: `'/dp-advanced-iterative-methods/'` pro následující url.

https://veselj43.github.io/dp-advanced-iterative-methods/

Kromě konfigurace už se jedná jen o sérii příkazů spuštěných v adresáři `impl`.

- Instalace (resp. stažení) závislostí `npm install` (lze spustit před úpravou konfigurace)
- Sestavení `npm run build`
- Výsledné soubory jsou k nalezení v adresáři `impl/dist/`
- Pro nasazení je zapotřebí zkopírovat obsah adresáře `impl/dist/` na server.

Jakmile jsou soubory sestavené s odpovídající konfigurací přístupné, aplikace je nasazena.

7.2 Požadavky na server a bezpečnost

Požadavky na server jsou minimální, skoro by se dalo říci žádné. Výsledkem produkčního sestavení jsou soubory čistě v HTML, CSS a JS. Tyto soubory stačí jen publikovat na požadované adrese a vše bude fungovat. Veškerý kód totiž interpretuje až samotný prohlížeč u klienta.

S tím souvisí také bezpečnost aplikace. Reálně aplikace běží pouze na straně klienta a server nemusí vědět o žádné akci uživatele kromě prvotního načtení zdrojových souborů. Aplikace opravdu neposílá žádná jiná data na server, než samotný požadavek na načtení požadované url. Díky tomu jsme se vyvarovali řadě běžných bezpečnostních rizik. Například jakákoliv varianta *code injection*. Také nejsou vyvolány žádné dotazy na jiné webové služby, vše se děje lokálně na straně klienta. Výhodou je minimální požadavek na výkon a prostředí serveru, téměř neomezená škálovatelnost, a v podstatě implicitní ochrana proti útokům vztahujícím se ke kódu aplikace.

Kapitola 8

Rozdělení práce

Finální aplikace vznikla v rámci tří diplomových prací. Každý ve své práci přikládáme tabulku 8.1. Tato tabulka slouží jako přehled rámcového rozdělení implementace jednotlivých částí aplikace.

Tabulka 8.1: Rámcový přehled rozdělení práce na aplikaci

	Kluzáček M.	Kugler A.	Veselý J.
Návrh UI	✓	konzultant	konzultant
Implementace prototypu	konzultant	konzultant	✓
Testování prototypu	asistent	✓	asistent
Implementace společného UI	konzultant	konzultant	✓
Simulované ochlazování	✓		
Genetický algoritmus		✓	
Tabu prohledávání			✓
Graf průběhu		GA modifikace	✓
SAT	generátor		problém
TSP	✓		
Batoh	✓		
MVC	✓		
ETSP		✓	
Závěrečné testování	SO	GA	Tabu
Nasazení aplikace			✓

Závěr

Cílem práce bylo vytvořit aplikaci demonstrující Tabu prohledávání pro výuku předmětu MI-PAA.

Osobně si myslím, že zadání bylo splněno. Aplikace má přehledné uživatelské rozhraní a je velice snadno přístupná. Aktuální verze obsahuje pět problémů včetně generátoru instancí. Tabu prohledávání má možnost nastavení krátkodobé a dlouhodobé paměti, počtu iterací, rozsahu prohledávání okolí a výběr mezi výchozím a náhodným startovním stavem. Je k dispozici grafické sledování běhu a porovnávání dokončených běhů. Ke každému běhu jsou také k dispozici parametry, se kterými byl algoritmus spuštěn, parametry instance, čas běhu a nejlepší nalezený stav (hodnota cenové funkce a zobrazení konfigurace).

Konkrétně jsem vytvořil prototyp a následně implementoval UI pro společnou část aplikace a pro Tabu. Implementoval jsem problém SAT a algoritmus Tabu prohledávání. Nejprve jsem navrhl a následně optimalizoval proces vizualizace pomocí grafu a komunikace hlavního vlákna (UI) s výpočetním (Web Worker). Implementoval jsem ukládání stavu aplikace do localStorage a instancí s výsledky běhů do IndexedDB.

Výsledky testování odhalily nedostatky a pomohly je opravit. Důležitým poznatkem je, že se nejednalo o chyby, které by indikovaly nesplnění požadavků na aplikaci a tím cíl práce. Velmi hodnotná je také pozitivní zpětná vazba od studentů, kteří aplikaci testovali.

Do budoucna je možné přidat další problémy, případně i algoritmy, díky univerzálnímu rozhraní. Vybrané technologie jsou v současné době populární, některé poměrně nové (již ale spolehlivě fungují), a lze předpokládat rozumně dlouhou životnost aplikace.

Doufám, že bude v následujících ročnících reálně používána a pomůže jak studentům k pochopení algoritmů, tak vyučujícím k vysvětlení látky.

Literatura

[1] Kluzáček, M.: *Aplikace pro demonstraci funkce simulovaného ochlazování.* Diplomová práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2018.

[2] Kugler, A.: *Aplikace pro demonstraci funkce genetických algoritmů.* Diplomová práce, České vysoké učení technické v Praze, Fakulta informačních technologií, Praha, 2018.

[3] Glover, F.: Future paths for integer programming and links to artificial intelligence. In *Computers & Operations Research*, ročník 13, 1986, ISSN 03050548, s. 533–549, doi:10.1016/0305-0548(86)90048-1, [cit. 2018-03-25]. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/0305054886900481

[4] Gendreau, M.; Potvin, J.-Y.: Tabu search [online]. [cit. 2018-03-25]. Dostupné z: https://pdfs.semanticscholar.org/9143/fc08cb934f77e9788adab22ac4b386c7b5f5.pdf

[5] Glover, F.; Martí, R.: Tabu search [online]. [cit. 2018-03-25]. Dostupné z: https://www.uv.es/~rmarti/paper/docs/ts2.pdf

[6] Glover, F.; LAGUNA, M.: Principles of Tabu Search [online]. [cit. 2018-03-25]. Dostupné z: https://www.uv.es/~rmarti/paper/docs/ts1.pdf

[7] Glover, F.; LAGUNA, M.; Martí, R.: Principles and Strategies of Tabu Search [online]. [cit. 2018-03-25]. Dostupné z: https://www.uv.es/~rmarti/paper/docs/ts3.pdf

[8] Browser Market Share [online]. 2017, [cit. 2018-03-11]. Dostupné z: https://netmarketshare.com/browser-market-share.aspx

[9] WHATWG: *HTML Living Standard [online]*. [cit. 2018-03-11]. Dostupné z: https://html.spec.whatwg.org/#workers

[10] What is wireframing? [online]. 2018, [cit. 2018-04-01]. Dostupné z: https://www.experienceux.co.uk/faqs/what-is-wireframing/

[11] Vue.js [online]. [cit. 2018-04-02]. Dostupné z: https://vuejs.org/

[12] What is Vuex? [online]. [cit. 2018-04-02]. Dostupné z: https://vuex.vuejs.org/en/intro.html

[13] ZangoDB [online]. [cit. 2018-04-02]. Dostupné z: https://erikolson186.github.io/zangodb/

[14] dc.js - Dimensional Charting Javascript Library [online]. [cit. 2018-04-02]. Dostupné z: https://dc-js.github.io/dc.js/

[15] Bootstrap [online]. [cit. 2018-04-02]. Dostupné z: https://getbootstrap.com/docs/3.3/

[16] Bootstrap 3 Components implemented by Vue 2 [online]. [cit. 2018-04-02]. Dostupné z: https://uiv.wxsm.space/

[17] Canvas vs. SVG: Choosing the Right Tool for the Job [online]. [cit. 2018-04-13]. Dostupné z: https://www.sitepoint.com/canvas-vs-svg-choosing-the-right-tool-for-the-job/

[18] SVG, nebo Canvas? Vyberte si [online]. [cit. 2018-04-13]. Dostupné z: https://www.zdrojak.cz/clanky/svg-nebo-canvas-vyberte-si/

[19] WWW Consorcium: *Scalable Vector Graphics (SVG) 1.1 (Second Edition) [online]*. [cit. 2018-03-25]. Dostupné z: https://www.w3.org/TR/2011/REC-SVG11-20110816/

[20] Webpack [online]. [cit. 2018-04-22]. Dostupné z: https://webpack.js.org/

Příloha **A**

Seznam použitých zkratek

PRNG Pseudo Random Number Generator

HTML HyperText Markup Language

CSS Cascading Style Sheet

SVG Scalable Vector Graphics

JS JavaScript

UI User Interface

UX User Experience

lo-fi Low Fidelity

hi-fi High Fidelity

Příloha **B**

Obsah přiloženého DVD

```
readme.txt ..... stručný popis obsahu DVD
DP_Vesely_Jaroslav_2018.pdf ..... text práce ve formátu PDF
src
    thesis ..... zdrojová forma práce ve formátu LaTeX
    impl ..... zdrojové kódy implementace
```