VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

MOBILNÍ APLIKACE PRO ROZVRHOVANÍ DISCIPLÍN NA DĚTSKÉM TÁBOŘE

MOBILE APPLICATION FOR SCHEDULING ACTIVITIES AT SUMMER CAMPS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ONDŘEJ HOLUB**
AUTHOR

VEDOUCÍ PRÁCE — **RNDr. MILAN ČEŠKA, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Holub Ondřej**
Program: Informační technologie
Název: **Mobilní aplikace pro rozvrhovaní disciplín na dětském táboře**
Mobile Application for Scheduling Activities at Summer Camps
Kategorie: Informační systémy

Zadání:

1. Seznamte se se současnými přístupy pro tvorbu rozvrhovacích systémů včetně jejich mobilních a webových rozhraní.
2. Seznamte se s optimalizačními a rozvrhovacími algoritmy pro minimalizaci konfliktů a vyberte z nich nejvhodnější pro použití v kontextu rozvrhování disciplín na dětském táboře.
3. Navrhněte a implementujte výslednou mobilní rozvrhovací aplikaci, dbejte přitom na maximální jednoduchost používání koncovým uživatelem.
4. Navrhněte a implementujte rozšíření aplikace dovolující interaktivně přeplánovat existující rozvrh dle nových požadavků a omezení.
5. Otestujte funkcionalitu aplikace na vhodných netriviálních datech. Vyhodnoťte užitečnost aplikace z pohledu koncového uživatele.

Literatura:

- Ch. Artigues, S. Demassey, and E. Neron. 2007. *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*. ISTE.
- E. Andersson, P. Greenspun, and A. Grumet. 2006. Software Engineering for Internet Applications. The MIT Press.
- E. Kangas and T. Kinnunen. "Applying User-Centered Design to Mobile Application Development." *Commun ACM* 48, no. 7 (2005): 55-59.

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání a začátek prací na bodě třetím.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Češka Milan, RNDr., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 14. května 2020
Datum schválení: 31. října 2019

Abstrakt

Práce se zabývá návrhem a tvorbou mobilní aplikace usnadňující plánování aktivit na dětském táboře na základě preferencí účastníků. Aplikace je implementována způsobem, který zajišťuje minimalizaci konfliktů preferencí s ohledem na dodatečná, uživatelem definovaná omezení. Pro efektivní zadávání vstupních dat je využito skenování karet účastníků, které obsahují QR kódy. Samotný proces hledání optimálního rozdělení je formalizován jako problém barvení grafu a je implementován s využitím metody větví a mezí. Vytvořený systém nevyžaduje pro své fungování žádné další zařízení, veškerý výpočet běží přímo na mobilním telefonu uživatele. Díky systému byl významně urychlen a zjednodušen proces plánování rozvrhů na dětských táborech.

Abstract

The main goal of this thesis is to design and implement a mobile application assisting in scheduling activities at summer camps taking into account the preferences of camp participants. This is done in a way that ensures the minimization of conflicts with respect to user-defined constraints. QR code scanning is used as an effective method of data input. The main process of finding an optimal solution is formalized as a graph colouring problem and is implemented using the branch and bound method. The created system does not require any other device for its functioning, all the necessary calculations are being executed on a mobile phone of the user. This system has significantly accelerated and simplified the process of scheduling activities at summer camps.

Klíčová slova

mobilní aplikace, plánování s omezeními, minimalizace konfliktů, problém barvení grafů, metoda větví a mezí, efektivní zadávání vstupních dat, Flutter, QR, metoda větví a mezí, operační analýza

Keywords

mobile application, constraint scheduling, conflict minimization, graph colouring problem, branch and bound method, effective input of data, Flutter, QR, Branch and Bound method, operational analysis

Citace

HOLUB, Ondřej. *Mobilní aplikace pro rozvrhovaní disciplín na dětském táboře*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce RNDr. Milan Češka, Ph.D.

Mobilní aplikace pro rozvrhovaní disciplín na dětském táboře

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana RNDr. Milana Češky, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Ondřej Holub
25. května 2020

Poděkování

Rád bych poděkoval svému vedoucímu, panu RNDr. Milanu Češkovi, Ph.D. za vedení a věcné připomínky, které mi pomohly při psaní této práce. Kromě toho bych chtěl poděkovat svým rodičům za podporu, rady a jazykovou korekturu, byli jste mi opravdovou oporou. Dále bych chtěl poděkovat panu Lukáši Holubovi a slečně Romaně Fridrichové, kteří mi pomohli zrealizovat uživatelské testování. Nakonec nesmím vynechat ani Cirkus LeGrando, díky kterému se má závěrečná práce bude používat v praxi.

Obsah

Kapitola 1

Úvod

Široká dostupnost výpočetní techniky v mnoha podobách umožňuje její nasazení na řadu plánovacích úloh, jejichž řešení je s využitím čistě lidských zdrojů velmi náročné. Jednou z těchto plánovacích úloh je i úloha hledání optimálního rozdělení disciplín. Zmíněná úloha se zabývá nalezením takového sloučení aktivit (disciplín) do několika bloků (rund), aby byl minimalizován počet zájemců o účast v aktivitách, které byly zařazeny do stejného bloku. Při přiřazování navíc musí být respektována i další, dle situace definovaná omezení. Řešení obdobných plánovacích úloh na základě známých omezení (obecně známých jako Constraint-Based Scheduling Problems) zahrnuje operační analýzu, tedy vytvoření modelu a jeho následné využití optimalizačními metodami. Výběr vhodného modelu a metody je zásadní, neboť různé metody mohou řešit problém rozdílně efektivními přístupy a jednotlivé modely mohou mít limitace s ohledem na daný problém. Nalezení fungující kombinace modelu a metody bude pro řešení problému rozdělení klíčové.

Kategorie plánovacích úloh je dobře známa, spadají do ní problémy z reálného světa, a to například z oboru logistiky (Integrated Transportation Problem [14]), výroby (Scheduling for Cellular Manufacturing [10]), skladového hospodářství (Optimal inventory policy within hospital space constraints [12]), námořnictva (Irish Navy Maintenance Scheduling [11]) a mnoha dalších. Řešení takovýchto úloh bývá komplexní a výpočetně náročné, je však klíčem k automatizaci a zefektivňování procesů, které byly dosud řešeny mnohdy z velké části kombinací intuice a metody pokus-omyl. Užití operační analýzy může vést k prokazatelně optimálním výsledkům, a to ve zlomku času a lidských zdrojů nutných k fungování předchozích metod.

Cílem této práce je vytvořit systém usnadňující řešení problému optimálního rozdělení disciplín s garancí minimálního počtu konfliktů, a to v jednotkách až desítkách minut (v současné době vyžaduje hledání rozdělení tým 4 lidí a průměrně 40 minut času, přičemž není garantována optimálnost nalezeného řešení). V ideálním případě bude systém využívat pouze jediného mobilního zařízení, a bude jej tedy možné použít téměř v jakýchkoliv podmínkách.

Kapitola 2 se zabývá detailním popisem problému, včetně jeho dosavadního řešení. Ve 3. kapitole je zpracován návrh algoritmu použitého pro hledání optimálního rozdělení. Kapitola 4 se zabývá návrhem systému pro řešení problému; kromě uživatelského rozhraní je zde rozebrána i metoda pro efektivní zadávání dat. V 5. kapitole jsou představeny použité technologie a popsány zajímavé implementační aspekty mobilní aplikace. 6. kapitola popisuje uživatelské testování a zaobírá se i otázkou použitelnosti aplikace pro větší vzorky vstupních dat. V 7. kapitole je nastíněn směr další možné práce v případě, že by bylo nutné aplikaci rozšiřovat. Také je zde přítomno závěrečné shrnutí.

Kapitola 2

Problém hledání optimálního rozdělení disciplín

Problém hledání optimálního rozdělení disciplín, jímž se tato práce zabývá, je několikrát ročně řešen na příměstských táborech Cirkusu LeGrando. První den tábora mají účastníci možnost vybrat si disciplíny, které by rádi po zbytek tábora trénovali. Disciplíny je pak nutné sloučit do tří rund určitým způsobem. *Runda* je v této práci definována jako množina disciplín, které probíhají souběžně. Existence dvou disciplín vybraných jedním účastníkem ve stejné rundě by znamenala, že účastník by musel být přítomen na dvou místech současně, a tato situace je definována jako *konflikt*. Nalezení rozdělení s nulovým počtem konfliktů bývá netriviální, v mnoha případech dokonce nemožné - důvod bude prezentován později v této kapitole. Pokud nebude možno nalézt rozdělení s nulovým počtem konfliktů, je nezbytné, aby někteří účastníci změnili své preference. Problém hledání optimálního rozdělení disciplín se zabývá i těmito změnami. Obecně problém spadá do kategorie optimalizačních úloh, které se zaměřují na nalezení co nejoptimálnějšího řešení a které je možné řešit širokou škálou metod, souhrnně známých jako operační analýza. Při použití operační analýzy je nejdříve vytvořen model situace, na kterém je následně provedena optimalizace - tímto způsobem je řešen i problém optimálního rozdělení disciplín. Ze stejné kategorie problémů je známý například Nurse Scheduling Problem [1].

2.1 Popis problému

Existuje určitý počet disciplín (obvykle 16) a určitý počet účastníků (obvykle 60), konkrétní počty se mezi jednotlivými turnusy táborů mohou měnit. Každý účastník si vybírá právě dvě disciplíny. Dále existují omezení (také se mohou měnit mezi turnusy), která je možné rozdělit do dvou kategorií:

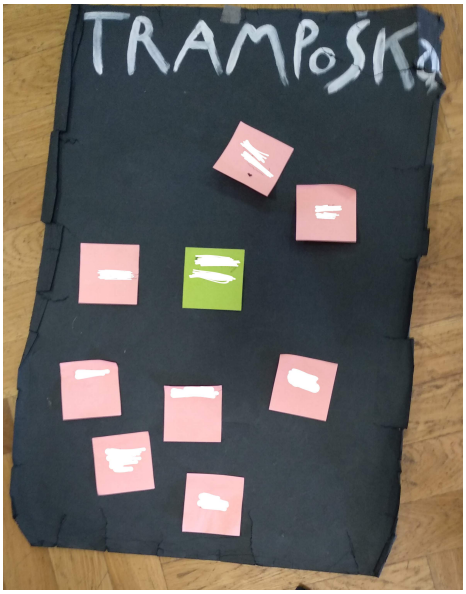
- Omezení počtu dětí na disciplíně (kapacitní důvody)
- Omezení prezence dvojice disciplín v jedné rundě (různé disciplíny mohou vyžadovat stejné vybavení, jehož je omezený počet)

Řešením problému je rozdělení disciplín do tří rund způsobem, kdy je počet konfliktů nulový a zároveň jsou dodržena všechna omezení. Pokud existuje více možných řešení, pak je vhodné nechat výběr finálního řešení na řešiteli a poskytnout mu všechna nalezená řešení se stejným, minimálním počtem konfliktů.

Důvodem, proč se disciplíny rozdělují právě do tří rund, je lety ověřená rovnováha časového harmonogramu účastníků - dva bloky aktivity, jeden blok volného času. Více rund (a tedy kratší intervaly) vede k nedostatku tréninkového času, méně rund (a tedy delší intervaly) obvykle vedou ke ztrátě soustředění dětí.

V průběhu hledání rozdělení může nastat situace, kdy neexistuje řešení s nulovým počtem konfliktů (důkaz, že tato situace může nastat, bude rozebrán v sekci 2.3). V takovém případě je nutné určit účastníky, kteří si musí změnit volbu jedné ze svých vybraných disciplín. Pro zajištění co největší spokojenosti účastníků je třeba zajistit, aby bylo tímto způsobem nutné provést co nejmenší počet změn - je tedy snaha o minimalizaci počtu konfliktů.

2.2 Dosavadní řešení problému



Obrázek 2.1: Papír reprezentující disciplínu "trampolína"s připevněnými papírky obsahující jména zájemců o trénink této disciplíny. Jména jsou z právních důvodů skryta.

Dosud se problém hledání optimálního rozdělení disciplín řešil experimentální metodou, která je časově náročná a nezajišťuje nalezení optimálního řešení (řešení s minimálním počtem konfliktů).

Každý z účastníků obdržel dva různobarevné (barvy nazvěme A a B) lepivé papírky a na každý z nich napsal své jméno. Tyto papírky pak daný účastník nalepil na velké papíry nadepsané názvy disciplín, jež papíry reprezentovaly (jak je vidět na obrázku 2.1). Nalepení papírku se jménem tedy znamenalo volbu dané disciplíny. Vedoucí tábora pak do první rundy přiřadili papíry disciplín na nichž byly papírky se jmény s převažující barvou A, do druhé rundy papíry s převažující barvou B a do třetí zbytek. Následně se postupně vybíraly disciplíny nesplňující omezení (případně mající vysoký počet konfliktů) a každá z nich byla přeřazena (případně vyměněna) v případě, že tato změna způsobila snížení celkového počtu konfliktů. Celý proces hledání optimálního uspořádání může připomínat fungování algoritmu Min-conflicts [13], avšak s ohledem na lidský faktor mohly nastat při výměnách jisté nedokonalosti a objevovaly se zde prvky chaotického chování. Kvalita výsledku tedy

mnohdy záležela na kombinaci náhody a vytrvalosti instruktorů. Pokud uspořádání získané tímto způsobem obsahovalo nenulový počet konfliktů, bylo možné dokázat existenci lepšího řešení pouze nalezením takového řešení.

2.3 Formalizace problému

Pro nalezení vhodné metody řešení problému hledání optimálního rozdělení disciplín je dobré se na problém podívat z matematického hlediska a jeho jednotlivé komponenty řádně definovat. Nejprve je nutno definovat systém výběru disciplín účastníky tábora.

Existuje konečná množina D obsahující nabízené disciplíny.

$$D = \{D_1, D_2, ..., D_m\}$$

Existuje konečná množina V dvojic disciplín z množiny D. Každá dvojice značí disciplíny vybrané jedním účastníkem.

$$V = \{V_1, V_2, ..., V_n\}, V_1 = (D_1, D_2)$$

Systém výběru disciplín je tedy možné definovat jako dvojici S.

$$S = (D, V)$$

Dále je nutné definovat, co znamená příslušnost disciplíny rundě. Pokud jsou rundám přiřazena čísla (*1*, *2*, *3*), pak je možné se dívat na tuto příslušnost jako přiřazení čísla rundy dané disciplíně, jinými slovy se jedná o zobrazení p.

$$p : D \mapsto \{1, 2, 3\}$$

Nyní je možné definovat, co znamená konflikt. Jedná se o situaci, kdy obě disciplíny vybrané jedním účastníkem přísluší stejné rundě. Aby se zamezilo přítomnosti konfliktů, je třeba říci, že nesmí nastat situace, kdy se oba prvky množiny D náležící jedné dvojici množiny V zobrazují do stejného čísla rundy.

$$\forall (V_1, V_2) \in V : p(V_1) \neq p(V_2)$$

Je možné si všimnout, že systém výběru disciplín S odpovídá definici grafu [6], kde množina disciplín D odpovídá množině vrcholů grafu a množina dvojic výběru V odpovídá množině hran grafu. Zároveň definice příslušnosti disciplíny k rundě společně s definicí konfliktu odpovídá definici problému barvení grafu [7].

Před zkoumáním známých vlastností barvení grafu je ještě nutné vzít v potaz omezení, která byla definována v sekci 2.1.

- Kapacitní omezení disciplíny je možné vnímat jako maximální počet hran (včetně jejich vah) vedoucí z jednoho vrcholu grafu; toto omezení bude nutné brát v potaz během následující práce.
- Omezení prezence dvou disciplín v jedné rundě lze implementovat vytvořením hrany speciálního typu (abychom předešli její detekci při počítání kapacitního omezení) s váhou odpovídající prioritě omezení. Váha takového omezení bude rovna nějakému vysokému číslu (například součtu počtu účastníků a priority daného omezení). Tímto se docílí kýženého stavu, kdy není za žádné okolnosti řešení považováno za optimální, je-li takovéto omezení porušeno. Teoreticky by mohlo být rozdělení s porušeným omezením považováno za optimální v případě, kdy speciální hrany utvoří v grafu kliku, a je tedy nutné tuto skutečnost brát v potaz během návrhu a implementace aplikace.

2.4 Alternativní modelování problému

Pro modelování problému jsem se rozhodl využít metodu barvení grafu kvůli její přímočarosti a jednoduchosti. Zkoumat různé přístupy je však důležité, jelikož modelování rozdílnými způsoby může mít pro výpočet rozdílné nároky - ať již na paměť, výpočetní náročnost či složitost implementace. Modelovat problém pomocí jiného přístupu by umožnilo jeho řešení například v případě, kdy se původní přístup ukáže jako výpočetně příliš náročný. V této sekci tedy rozeberu alternativní přístupy, pomocí kterých by bylo možné modelovat a následně řešit problém optimálního rozdělení disciplín.

Problém splnitelnosti booleovské formule (SAT)

Problém by bylo možné modelovat pomocí booleovské formule, pro jejíž ověřitelnost existuje řada efektivních nástrojů (kterých pro své fungování využívá i například jazyk Prolog). Tento přístup má však jeden podstatný problém. Umožňuje sice rychlé ověření, zda je vygenerovaná formule splnitelná, či nikoliv, neposkytuje však podporu pro minimalizaci konfliktů v případě nesplnitelnosti. Právě minimalizace konfliktů je významnou součástí řešení problému hledání optimálního rozdělení disciplín. Pro tento nedostatek jsem se rozhodl metodu modelování pomocí booleovské formule zavrhnout.

Celočíselné programování (Integer Programming)

Metoda celočíselného programování spočívá ve stanovení soustavy rovnic skládajících se z různých stavových proměnných reprezentujících data, která mají být optimalizována. Obvykle existuje speciální proměnná definovaná jednou rovnicí skládající se z velkého množství proměnných, kterou je třeba minimalizovat či maximalizovat. Samotný výpočet spočívá v přiřazování celočíselných hodnot stavovým proměnným a ve sledování hodnoty minimalizované/maximalizované speciální proměnné.

Problém optimálního rozdělení disciplín by se dal touto metodou modelovat následujícím způsobem.

Nechť $D_{i,r}$ označuje disciplínu D, kde i je identifikátor disciplíny a r je příslušnost disciplíny k rundě. Dále je stanoveno, že $D_{i,r}$ může nabývat pouze hodnot 1 nebo 0 (takže pracujeme s podmnožinou metody celočíselného programování, a to s booleovským programováním). Skutečnost, že každá disciplína náleží právě jedné rundě, je pak možné modelovat jako:

$$D_{i,1} + D_{i,2} + D_{i,3} = 1$$

Dále je nutné vytvořit proměnnou, jejíž hodnota bude sledována. V problému optimálního rozdělení je snaha o redukci konfliktů, tedy tato proměnná bude rovna součtu všech konfliktů a bude snaha ji minimalizovat. Konflikt nastává, pokud dvě disciplíny zvolené stejným účastníkem sdílí rundu, přičemž nezáleží na tom, o kterou rundu jde. Proto je třeba nejprve definovat konflikt K_i, kde i je identifikátor účastníka majícího zapsané disciplíny s identifikátory 1 a 2.

$$K_i = D_{a,1} * D_{b,1} + D_{a,2} * D_{b,2} + D_{a,3} * D_{b,3}$$

Pokud tedy budou disciplíny s identifikátory a a b sdílet rundu, bude hodnota proměnné K_i 1. V opačném případě 0.

K úplnosti je třeba ještě modelovat omezení. Omezení se skládá ze dvou disciplín, které nemohou sdílet rundu, takže je možné použít definici konfliktu a znásobit jeho hodnotu (bude postačovat počtem účastníků n) v případě, že dvě disciplíny a a b sdílí rundu:

$$O_i = (D_{a,1} * D_{b,1} + D_{a,2} * D_{b,2} + D_{a,3} * D_{b,3}) * n$$

Proměnná, kterou tedy bude snaha minimalizovat s ohledem na počet účastníků n, jejich volby a počet omezení m a jimi omezené disciplíny, bude definována jako:

$$M = \sum_{i=1}^{n} K_i + \sum_{j=1}^{m} O_j$$

Řešením problému tedy budou disciplíny s identifikátorem i náležící rundám r takové, kde $D_{i,r} = 1$ po minimalizaci proměnné M.

Tento způsob jsem však nevybral, jelikož se mi jeví modelování problému hledání optimálního rozdělení disciplín pomocí barvení grafu jako mnohem elegantnější a přímočařejší. Právě absence abstrakční vrstvy zjednodušuje zahazování určitých stavů, které je popsáno v kapitole 3, čímž může být významně usnadněn výpočet.

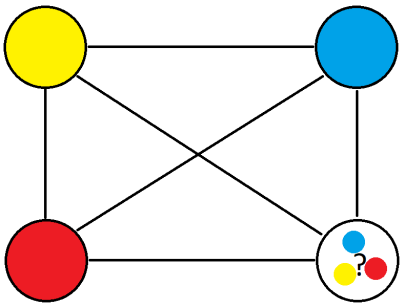
2.5 Barvení grafu třemi barvami a některé jeho vlastnosti

Vzhledem k tomu, že jsou disciplíny shlukovány do právě tří rund, je možné problém hledání optimálního rozdělení disciplín považovat za problém barvení grafu s chromatickým číslem $\chi = 3$ (neboli barvení grafu třemi barvami). O tomto barvení jsou známy následující, pro tuto práci podstatné, vlastnosti.

- Chromatické číslo grafu $\chi(G)$ musí být větší nebo rovno stupni největší kliky v grafu $\omega(G)$.

  $$\chi(G) \geq \omega(G)$$

  Jinými slovy, graf není možné obarvit třemi barvami, pokud obsahuje kliku alespoň čtvrtého stupně (jak je vidět na obrázku 2.2). Absence takových klik však není důkazem, že je graf obarvitelný třemi barvami [2]. Pro problém hledání optimálního rozdělení disciplín to znamená, že může nastat situace, kdy není možné rozdělit disciplíny do tří rund s nulovým počtem konfliktů.



Obrázek 2.2: Kompletní graf o čtyřech vrcholech - klika čtvrtého stupně. Vrcholům takového grafu nelze přiřadit jednu ze tří hodnot způsobem, kdy je každým dvěma vrcholům sdílejících hranu přiřazena rozdílná hodnota.

- Nalezení obarvení grafu třemi barvami či důkaz, že takovéto obarvení neexistuje, spadá do kategorie NP-složitých úloh [9]. Pro důkaz nemožnosti obarvitelnosti grafu třemi barvami je nutné dle dosud získaných znalostí o NP třídě problémů prohledat všechna možná řešení. To samé platí i pro řešení problému hledání optimálního rozdělení disciplín, jelikož se jedná pouze o jinak formulovaný problém barvení grafu. Je také nutné brát v potaz požadavek na provedení finálního výběru mezi řešeními se stejným počtem konfliktů vedoucími tábora (jak bylo určeno v sekci 2.1). Z toho vyplývá, že ani nalezení řešení s nulovým počtem konfliktů neznamená konec hledání, k instruktorovi se musí dostat všechna řešení s minimálním počtem konfliktů. Hledání je však možné zefektivnit vhodným algoritmem, který bude prezentován v následující kapitole.

Metoda barvení grafu je běžně používána právě pro řešení optimalizačních problémů. Konkrétně byla například použita pro plánování přidělování registrů [3].

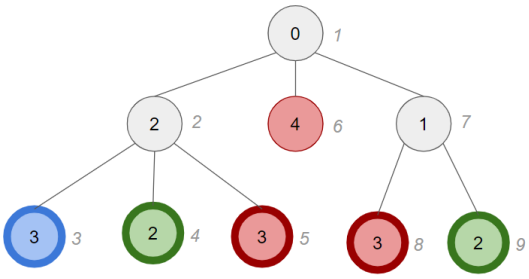
Kapitola 3

Algoritmus pro hledání optimálního rozdělení disciplín

Z požadavku (uvedeného v sekci 2.1) na nalezení všech existujících řešení s minimálním počtem konfliktů plyne, že je pro řešení tohoto problému nutné použít algoritmu schopného nalézt globální minimum. Také je vhodné vybrat algoritmus, který neprohledává celý stavový prostor, ale používá heuristiku a zaměřuje se pouze na stavy vedoucí ke stejnému či lepšímu výsledku. K tomu by mohla být nápomocná skutečnost, že je poměrně jednoduché spočítat počet konfliktů. Každý účastník je totiž konfliktem, pokud jsou obě jeho disciplíny ve stejné rundě. V ideálním případě by byl algoritmus spouštěn přímo na mobilním zařízení, jehož bude využito pro načítání vstupních dat. Limitem při výpočtu by mohla být jak paměť, tak nedostatečný výkon mobilního zařízení. Výpočet s nedostatečnou pamětí by mohl vést k nutnosti zahazování některých stavů, a tím pádem k neoptimálním řešením. Výpočet s nízkým výkonem by se projevil pouze dlouhým časem běhu.

3.1 Obecný algoritmus Branch and Bound (metoda větví a mezí)

Při hledání vhodného algoritmu jsem se rozhodl vyzkoušet metodu větví a mezí (známou jako B&B, Branch and Bound). Jedná se o elegantní algoritmus používaný právě pro optimalizační problémy, který prohledává stavový prostor, jako by se jednalo o strom. Kořenem se označuje počáteční (nulové) řešení, které se dále rozvětvuje. Každý uzel pak symbolizuje stav, dvojice uzlů oddělená jednou větví se liší změnou jedné proměnné. Při prohledávání stavového prostoru se pro každý uzel počítá minimální či maximální hodnota jeho podstromu (v případě problému rozdělení to bude minimální počet konfliktů), což umožňuje celý podstrom zahodit, je-li jeho minimální/maximální hodnota větší/menší než nejlepší dosud nalezená. Algoritmus tedy neprohledává celý stavový prostor, ale i tak garantuje nalezení globálního minima/maxima za předpokladu, že hodnota podstromu uzlu nikdy neklesá (v případě hledání minima), či neroste (v případě hledání maxima). Názornou vizualizaci fungování algoritmu poskytuje obrázek 3.1.



Obrázek 3.1: Demonstrace fungování algoritmu Branch and Bound. Vpravo od uzlu je kurzívou znázorněno jeho pořadí, číslo v uzlu udává jeho hodnotu (a tedy minimální hodnotu podstromu daného uzlu), listové uzly jsou zvýrazněny tučnými okraji. Modře jsou zvýrazněny uzly, jež byly považovány za minimum, ale byla nalezena lepší řešení. Zeleně jsou zvýrazněny uzly reprezentující optimální řešení. Červeně jsou zvýrazněny uzly, které byly společně se svými podstromy z prohledávání vyřazeny, neboť nevedly k optimálnímu řešení. Byla použita varianta prohledávání do hloubky.

Algoritmus má dvě obecné podoby, tzv. horlivou (Eager) a línou (Lazy) [4], každá z variant může být implementována pomocí fronty nebo zásobníku. Horlivá podoba se od líné liší pořadím funkce *branch*() a operace *bound*, zde reprezentované případným zahazováním uzlů ve smyčce *foreach*. Algoritmus 1 je jednou z variant algoritmu Branch and Bound minimalizujícího řešení funkce $f(x)$ (kde x je uzel obsahující řešení):

Algorithm 1: Horlivá varianta algoritmu Branch and Bound využívající zásobník

```
Min = ∞;
Stack = [initialSolution];
Best = null ;
while Stack is not empty do
    Node = Stack.pop();
    Children = branch(Node);
    foreach Child in Children do
        if Child is final AND f(Child) < Min then
            Min = f(Child);
            Best = Child;
        else
            if f(Child) < Min then
                Stack.push(Child);
            end
        end
    end
end
```

K tomuto algoritmu je vhodné připojit několik poznámek pro jeho lepší pochopení:

- Proměnná *Min* obsahuje dosud nejmenší nalezenou hodnotu $f(x)$.
- Zásobník *Stack* obsahuje dosud neprozkoumané stavy; při inicializaci je na něm uloženo pouze jedno, počáteční řešení. Žádná z proměnných tohoto řešení nemá přiřazenou hodnotu.

- Proměnná $Best$ obsahuje nejlepší nalezené řešení. Toto řešení je také výstupem algoritmu.
- Metoda $branch(Node)$ vytvoří z otcovského uzlu $Node$ množinu synovských uzlů $Children$. Každý z těchto uzlů nese unikátní řešení lišící se od řešení otcovského uzlu pouze hodnotou jedné proměnné.
- K potenciálnímu zahazování uzlů $Child$ dochází jejich neumístěním na zásobník $Stack$.

Po běhu algoritmu bude proměnná Min obsahovat minimální hodnotu $f(x)$ a proměnná $Best$ bude obsahovat jedno z nejlepších nalezených řešení.

Je třeba zvážit, zda pro algoritmus využít fronty či zásobníku. Algoritmy prohledávající stavový prostor využívající fronty bývají obvykle náročnější na paměť, v případě zásobníku mohou být pomalejší. Abychom efektivně využili vlastností algoritmu Branch and Bound, doporučuje se jeho použití se zásobníkem. Frontu je vhodné použít v případech, že je na začátku běhu použita heuristika pro určení počáteční hodnoty Min. Pokud by bylo použito původní nekonečno, zahazování uzlů by probíhalo až na listové úrovni, což je při použití s frontou velmi neefektivní. Bez počátečního odhadu, ale s pomocí zásobníku se algoritmus poměrně rychle dostane na úroveň listů a bude záležet na rozložení hodnot řešení, zda a jak často budou uzly zahazovány. Proto je vhodné použít variantu se zásobníkem.

U algoritmu je také nutné vybrat vhodnou funkci $branch()$ způsobem, aby se na různých místech prohledávaného stromu neobjevovaly uzly s identickými řešeními. Je také nutné zajistit, aby byla funkce $f(x)$, která je zodpovědná za ohodnocování uzlů, co nejrychlejší [4]. Právě těmito změnami se zabývá následující sekce.

3.2 Úprava algoritmu B&B k použití pro řešení problému rozdělení

Z obecné verze algoritmu Branch and Bound je vidět, že algoritmus jen minimalizuje hodnotu funkce a poskytuje pouze jedno řešení mající tuto hodnotu. Pro použití k řešení problému optimálního rozdělení disciplín je však nutné, aby algoritmus vracel všechna řešení mající tuto hodnotu. Také bude třeba navrhnout implementace funkcí $f(x)$ a $branch(Node)$ zmíněných v obecné podobě algoritmu Branch and Bound. Také je třeba zajistit, aby během větvení mohly konflikty pouze vznikat (nikoliv zanikat), což je základní předpoklad, aby mohla být některá řešení zahazována (a tedy aby mohl algoritmus správně fungovat).

Získávání všech řešení se stejným počtem konfliktů

Nejprve je nutné pozměnit proměnnou $Best$, nově bude obsahovat množinu všech nejlepších řešení. Pokud zkoumaný uzel $Node_i$ je koncovým řešením, pak bude jeho hodnota porovnána s hodnotou Min. Následně může nastat jedna ze tří situací:

1. $f(Node_i) > Min$; uzel $Node_i$ bude zahozen, neboť je horší než všechna dosud nalezená řešení uložená v proměnné $Best$.
2. $f(Node_i) = Min$; řešení uzlu $Node_i$ bude přidáno do množiny $Best$, neboť nalezené řešení je stejně dobré, jako jsou ta v množině $Node$.
3. $f(Node_i) < Min$; množina $Best$ bude nově obsahovat pouze řešení uzlu $Node_i$, předchozí obsah množiny bude zahozen, neboť se našlo nové a lepší řešení.

Díky této modifikaci bude po dokončení běhu algoritmu obsahovat proměnná *Best* všechna existující optimální řešení pro daný problém.

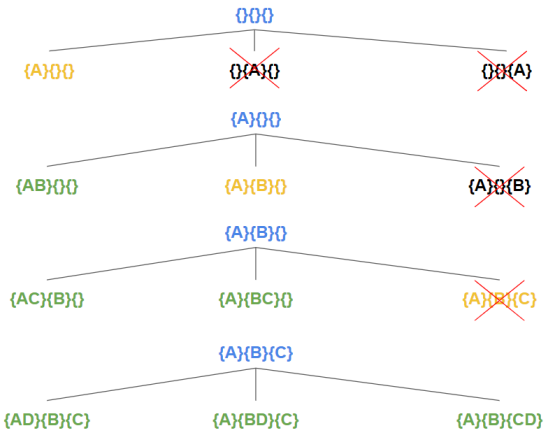
Návrh implementace funkce *branch*(*Node*) zajišťující průběh větvení

Obecná forma algoritmu Branch and Bound neudává, jakým způsobem má probíhat větvení řešení, jež zastupuje funkce *branch*(*Node*). Bylo již řečeno, že tato funkce musí produkovat řešení takovým způsobem, aby v rámci stromu nedocházelo k výskytu vícero uzlů nesoucích identická řešení a také aby konflikty nemohly zanikat. Pro zjednodušení nebude již pohlíženo na obecnou formu algoritmu, ale bude přizpůsoben pro použití k řešení problému optimálního rozdělení disciplín, který byl definován v sekci 2.1. Je již jasné, že každé řešení musí obsahovat tři množiny disciplín, pro každou rundu jednu. V počátečním stavu (kořeni stromu) budou tyto množiny prázdné. V sekci o obecném algoritmu B&B této kapitoly je uvedeno, že uzly oddělené jednou větví se liší změnou právě jedné proměnné. Vzhledem k tomu, že proměnné jsou množiny, měl by být rozdíl právě v jednom prvku právě jedné množiny. Je známa konečná množina disciplín, které je třeba rozdělit, takže by tento nárok mohl být splněn za dodržení postupu popsaného algoritmem 2, kterým je uzel *Parent* rozvětven na množinu uzlů *Children* (berouc v potaz definovaná omezení):

Algorithm 2: Návrh implementace metody branch(Node) zajišťující větvení uzlu

```
D = disciplines.removeFirst();
foreach Round in Parent.Rounds do
    if insertingDoesNotViolateLimitations(D, Round) then
        Child = clone(Parent);
        Child.insertDisciplineToRound(D, Round);
        Children.append(Child);
    end
end
```

Tento postup zajišťuje, že z uzlu *Parent* je vytvořeno tolik synovských uzlů *Child*, kolik existuje rund, pokud tedy nedojde k porušení omezení. Synovské uzly se vzájemně liší pouze rundou, do které byla vložena disciplína D. Také jsou disciplíny přidávány a nikdy nedojde k jejich odebrání, čímž je splněn požadavek na to, aby konflikty nemohly během hledání zanikat. Metoda *insertingDoesNotViolateLimitations* zajišťuje, že nedojde k takovému rozvětvení, kterým by došlo k narušení omezení. Po skončení této procedury budou tedy v množině *Children* uzly vytvořené z rodičovského uzlu *Parent*. Řešení v uzlech budou však unikátní, pouze pokud jsou rozlišovány množiny, reprezentující rundy, na instanční úrovni. Pokud by byly rozlišovány pouze podle svého obsahu, tedy pokud by platilo, že $(\{D_1\}, \{\}, \{\}) = (\{\}, \{D_1\}, \{\})$, řešení získaná tímto způsobem by nebyla unikátní. Unikátnost lze však zajistit lehkou úpravou výše uvedeného algoritmu: Pokud rodičovské řešení obsahuje identické rundy, pak v rámci jedné procedury větvení může být vložena disciplína do nejvýše jedné z těchto rund. Díky tomu budou získaná řešení unikátní. Zahazování uzlů při větvení je vizualizováno na obrázku 3.2.



Obrázek 3.2: Demonstrace různých případů větvení, kdy je povoleno pouze jedno vložení disciplíny do prázdné rundy za proceduru. Modře jsou označeny rodičovské uzly. Zeleně jsou označeny unikátní synovské uzly. Žlutě jsou označeny unikátní synovské uzly, které však vznikly vložením do prázdné rundy. Přeškrtnuty jsou synovské uzly, které byly zahozeny. Je vidět, že žluté a zahozené uzly jsou identické z hlediska obsahu rund, nezahození uzlu by znamenal výskyt dvou identických řešení ve stromu reprezentujícím stavový prostor. Identické rundy se tedy objevují, pouze pokud jsou disciplíny vkládány do dvou (nebo více) prázdných rund.

Každé řešení tedy bude muset nést informaci o tom, jaké disciplíny ještě nebyly přiřazeny do rund. Během větvení bude také nutné aktualizovat množinu účastníků v konfliktu, důvod a princip fungování bude představen v následující sekci.

Princip fungování funkce ohodnocující řešení, $f(x)$

Při návrhu ohodnocující funkce jsem vycházel ze dvou následujících faktů:

- Pokud jsou disciplíny přiřazovány do rund, mohou se nové konflikty pouze objevovat, nikoliv zanikat.
- Existující konflikty nebudou přiřazením libovolné disciplíny do libovolné rundy nijak ovlivněny.

Není tedy nutné přidáním disciplíny do rundy zjišťovat počet všech konfliktů, stačí pouze přičíst nově vzniklé konflikty k těm existujícím. Nebo ještě vhodněji, udržovat u každého řešení množinu účastníků, již jsou v konfliktu, a při vkládání disciplíny do rundy tuto množinu případně rozšířit. Počet prvků této množiny se pak bude rovnat počtu konfliktů. Existence množiny účastníků v konfliktu bude velmi užitečná při identifikaci těchto účastníků, jelikož bude každý z nich muset změnit jednu ze svých voleb (více v sekci 4.2).

Ohodnocující funkce $f(x)$ tedy bude pouze vracet počet prvků množiny účastníků v konfliktu, tato množina bude součástí každého řešení. Vkládání do množiny se bude dít během procedury větvení a bude fungovat na následujícím principu: Při větvení je známo, jaká disciplína se vkládá do jaké rundy, a stačí tedy zjistit, kteří účastníci vkládané disciplíny již mají zapsanou disciplínu této rundy, a vložit je do množiny účastníků v konfliktu. Ohodnocující funkce tak bude pracovat velmi rychle a bude výhodné využít kombinovanou variantu (horlivou i línou) algoritmu Branch and Bound.

Použití kombinované metody algoritmu

Horlivá a líná verze algoritmu se z výkonnostního hlediska liší v jedné zásadní věci. V případě horlivé metody je možné uzel, pokud splňuje dané podmínky, zahodit hned po tom, co byl větvením vytvořen ze svého rodiče. Tímto je ušetřeno místo v paměti. Problémem může být následující modelová situace: $Min = 10$, větvením vzniknou uzly A ($f(A) = 8$) a B($f(B) = 7$), oba jsou tedy ponechány a uloženy na zásobník. Následně algoritmus pokračuje uzlem B a dospěje k listovému uzlu s hodnotou 5. Vrátí se k uzlu A, který však již nezkoumá a rozvětví, i když je tato operace zbytečná, jelikož A nemůže vést k optimálnímu řešení, a mohl by být zahozen. Podstrom uzlu A je zahozen až po rozvětvení uzlu A. Protože je větvení (branch) výpočetně mnohem náročnější operací než ohodnocování a zahazování (bound), mohla by tato skutečnost negativně ovlivnit dobu výpočtu.

V případě líné metody jsou zase uchovávány na zásobníku i uzly, které by mohly být po rozvětvení ihned zahozeny a v případě vysokých stromů by to mohlo znamenat nezanedbatelné plýtvání pamětí.

Vzhledem k tomu, že je získání hodnoty počtu konfliktů a její porovnání s hodnotou Min opravdu rychlé, rozhodl jsem se obě metody zkombinovat. Případné zahazování řešení se tedy bude provádět jak po vyjmutí ze zásobníku řešení (před větvením), tak po větvení (kde budou případně zahazovány synovské uzly).

Kapitola 4

Návrh systému pro podporu řešení problému rozdělení

V této kapitole bude popsán návrh systému sloužícího jako nástroj pro řešení problému rozdělení. Primární důraz bude kladen na zajištění toho, aby systém bylo co nejjednodušší používat. S tím se pojí také nutnost jednoduchého a efektivního vkládání vstupních dat, zejména voleb účastníků. Neméně důležitou roli v usnadnění používání nástroje hraje také způsob, jakým jsou řazena optimální rozdělení, neboť mohou nastat situace, kdy je takových řešení nalezeno v řádu desítek či stovek. Vytvořený nástroj bude používán instruktory volnočasového Cirkusu LeGrando, na jehož příměstských táborech se problém několikrát ročně objevuje.

4.1 Vstupní data a způsob jejich zadávání

Nejprve je nutné položit si otázku, jaká vstupní data bude systém vyžadovat a jak je co nejvhodněji předat algoritmu hledajícímu optimální řešení problému. Pro určení těchto dat je možné využít shodnosti s problémem barvení vrcholů grafu. Pro barvení grafu je třeba znát:

- Chromatické číslo grafu, tedy kolika barvami je graf obarven.
- Vrcholy tvořící barvený graf
- Hrany spojující vrcholy barveného grafu

V předchozí sekci 2.3 bylo dokázáno, že problém optimálního rozdělení disciplín odpovídá problému barvení grafu za předpokladu, že:

- Každá barva odpovídá jedné rundě. Chromatické číslo grafu je rovno počtu rund, do kterých chceme disciplíny rozdělovat, $\chi = 3$.
- Vrcholy grafu (jež budou obarvovány) odpovídají disciplínám (jež budou rozdělovány do rund).
- Hrany spojující vrcholy grafu odpovídají účastníkům, kteří si vybrali dané disciplíny. Vzhledem k tomu, že jeden účastník si vybírá právě dvě disciplíny, odpovídá jedna hrana právě jednomu účastníkovi. Kromě toho mohou existovat i hrany reprezentující omezení - od hran účastníků se liší tím, že mají vyšší váhu (rovnou vysokému číslu, alespoň počtu účastníků).

K řešení problému optimálního rozdělení disciplín tedy potřebujeme znát disciplíny, které je třeba rozdělit, účastníky spolu s jejich volbami disciplín a seznam omezení (dvojic disciplín, které nemohou sdílet rundu). Kromě toho by neměla být opomenuta kapacitní omezení jednotlivých disciplín. Pro představu čtenáře o množství jednotlivých druhů dat je vhodné zmínit, že obvykle je třeba řešit problém pro 16 disciplín (každá má své vlastní, nekonkrétní kapacitní omezení), 60 účastníků (z čehož plyne 120 voleb, každý účastník má dvě) a 3 omezení (3 dvojice disciplín). Počty se zpravidla liší pouze v řádu jednotek. Nejnáročnější bude tedy zadat vstupní data o účastnících.

Při plánování načítání vstupních dat účastníků jsem se rozhodl využít již fungujícího, praxí ověřeného řešení: každá disciplína je reprezentována jedním velkým papírem. Každý účastník napíše své jméno na dva lepivé papírky a výběr disciplín provede přilepením papírku se jménem na papír nadepsaným názvem disciplíny, již reprezentuje. Co se děje dál, je popsáno v sekci 2.1.V průběhu vybírání, kdy jsou k disciplínám postupně připouštěny skupinky účastníků a mají možnost provést volbu jen jedné z disciplín, jsou kontrolována kapacitní omezení disciplín a případně se nábor do naplněné disciplíny uzavře. Obvykle se při naplnění kapacity dělají ústupky a kompromisy (je snaha vyjít účastníkům co nejvíce vstříc a kapacita je reálně omezena pouze schopností a ochotou instruktora dohlížet na daný počet účastníků), takže by nebylo vhodné ani žádoucí dodržovat kapacitní omezení striktně, neměnnou hranicí. Samozřejmě by bylo možné zadávat volby účastníků přímo do výpočetního zařízení, nebylo by však jednoduché brát v potaz kapacitní omezení, mohly by se tvořit fronty účastníků, vznikat chyby ze stran zadavatelů a opustil by se dosud používaný systém, který k těmto táborům přirostl natolik, že se stal jakýmsi samozřejmým ceremoniálem.

Cílem je tedy upravit toto fungující řešení způsobem, který by umožňoval snadnou konverzi dat o účastnících do digitální podoby. Jak již bylo zmíněno, jedná se o skupiny jmen účastníků (tedy unikátních identifikátorů) majících zájem o trénink jedné z nabízených disciplín. Nyní se nabízí položit si otázku: "Jaké metody se v praxi používají pro načtení velkého množství identifikátorů seskupených do různých skupin?"Odpověď se přímo nabízí, skenování kódů (čárových, RFID, QR...) v obchodech či skladech. Skenování kódů je v současné době velmi rychlé a v případě čárových či QR kódů jej jsou schopny provádět i mobilní telefony disponující kamerou a moderním operačním systémem - tedy v současné době téměř všechny. Pokud by každý účastník napsal své jméno na kartu s čárovým kódem a tuto kartu pak položil na již používaný papír nadepsaný názvem disciplíny, provedl by volbu velmi podobným způsobem, na jaký je zvyklý dosud. Proces zadávání dat do výpočetního zařízení by se pak mohl upravit do následujícího jednoduchého postupu. Nejprve instruktor zadá všechny disciplíny, u každé z nich naskenuje všechny kódy nacházející se na papíře nadepsaném touto disciplínou. Následně instruktor zadá všechna omezení ve formě dvojic disciplín.

Pro zadávání vstupních dat by tedy bylo vhodné vytvořit mobilní aplikaci. V ideálním případě by pak i realizovala výpočet hledání optimálního rozdělení disciplín, aby pro své fungování nevyžadovala připojení k internetu či jiný způsob komunikace se serverem, což by práci s ní mohlo dělat zbytečně složitou.

Návrh karet s kódy účastníků

Nejjednodušší by samozřejmě bylo používat čárové kódy (nejsou zdaleka tak komplexní jako QR). Je však žádoucí, aby se účastníci necítili jako kusy zboží v supermarketu (pocity by byly pravděpodobně následovány stížnostmi rodičů účastníků). Klasickým čárovým kódům

typu EAN-13 by tedy bylo lepší se vyhnout. Stále je však možné použít QR kódy, které jsou v současné době relativně populární a nejsou společností asociovány s obchodními artikly. Dnešní mobilní telefony jsou schopny načítat QR kódy velmi podobnou rychlostí (z pohledu běžného člověka) jako čárové kódy. Také je možné do QR kódu zakomponovat mnohem více dat, čehož sice nebude využito, mohlo by to však přijít k užitku při případné budoucí práci.

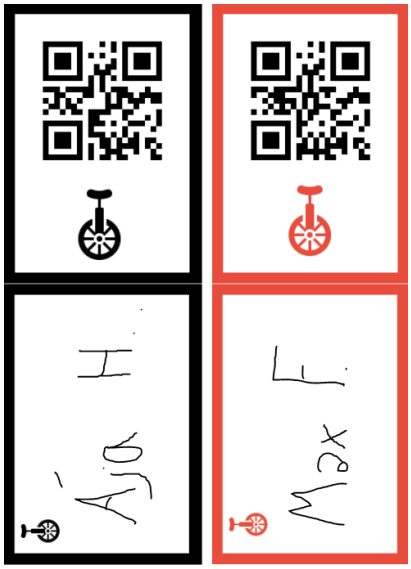
Součástí kódu by mohlo být přímo jméno účastníka, znamenalo by to nutnost tisknout každý příměstský tábor zhruba 120 nových karet účastníků (každý účastník potřebuje dvě identické karty, aby mohl provést dvě volby). Aby bylo předejito zbytečnému plýtvání papírem, je možné vytvořit univerzální karty s kódy, které se budou na každém táboře účastníkům propůjčovat. Kromě samotného kódu je tedy na kartě nutné umožnit účastníkům napsat své jméno (které bude před dalším použitím vymazáno), aby bylo možné jednoznačně přiřadit účastníka k jeho kódu. Karty bude nutné před každým rozdělením nutné zkompletovat po dvou, aby každý účastník dostal dvě karty s identickým kódem. Bylo by tedy vhodné, aby byly karty jednoduše identifikovatelné člověkem (označeny i jinak než QR kódem).

S ohledem na výše uvedené požadavky jsem vytvořil prototyp karet (viz obrázek 4.1). Na přední straně karty bude vždy QR kód se snadno rozpoznatelnou ikonou, na zadní straně stejná ikona a místo pro podpis účastníka. Každá karta bude zalaminovaná, aby se účastníci mohli podepsat lihovou fixou, která je snadno smazatelná pro opětovné použití karty. Celkem jsem takto vytvořil 24 karet s různými ikonami v černé variantě, pro dosažení dostatečného počtu 60 účastníků použiji stejný set ikon, ale rozdílných barev. Tímto způsobem se budou instruktorům karty snadněji kompletovat po dvou (barvu je možné identifikovat rychleji než tvar). Ikony byly použity ze serveru icons8[1] na základě bezplatné licence.

Každý z QR kódů nese informace o tvaru ikony (na prototypu "1kolka") a barvě karty účastníka (na prototypu "B"- black nebo "R"- red), oddělené znakem "_". Pro generování QR kódů bylo využito serveru the-qrcode-generator[2].

[1] https://icons8.com

[2] https://www.the-qrcode-generator.com/



Obrázek 4.1: Návrh karet, zobrazeny jsou dvě karty účastníka s identickou ikonou, ale v rozdílném barevném provedení. Je vidět přední i zadní strana každé karty. Na zadních stranách karet je naznačeno místo, kam se budou účastníci podepisovat. Ve verzi pro tisk tato jména nebudou.

Zadávání nabízených disciplín a omezení

V úvodu této sekce bylo naznačeno, že pro zadávání voleb účastníků by bylo nejvhodnější vytvořit mobilní aplikaci. Tuto aplikaci by bylo možné využít i pro zadávání nabízených disciplín a omezení. Pro disciplíny by jistě stačil vstupní formulář skládající se z textového pole, pro omezení by formulář obsahoval dvě tzv. "dropdown"tlačítka. Vzhledem k nízkému objemu dat (16 názvů disciplín a 2-3 omezení) se mi toto řešení jeví jako postačující.

4.2 Prezentace výsledků a podpora při výběru rozdělení

V kapitole 3 byl představen algoritmus Branch and Bound s úpravami, které umožní použití tohoto algoritmu pro řešení problému optimálního rozdělení disciplín. Výstupem algoritmu je množina všech řešení, která mají pro daný problém minimální počet konfliktů. Počet těchto řešení není nijak omezen (minimálně bude vždy alespoň jedno), záleží na vstupních datech. Z této množiny řešení bude muset instruktor pověřený nalezením vhodného rozdělení disciplín vybrat právě jedno, které bude použito pro aktuální turnus příměstského tábora. V kapitole 2.1 bylo již zmíněno, že výpočet má poskytnout seznam disciplín s minimálním počtem konfliktů, finální výběr však musí vždy provést člověk. Záleží totiž na velkém množství faktorů, jakými jsou například počet instruktorů na táboře, kvalifikace instruktorů vést příslušné disciplíny, případné spojování disciplín s malým počtem účastníků a další. Nicméně by bylo vhodné využít výpočetní techniku k tomu, aby byl instruktorovi výběr finální disciplíny co nejvíce usnadněn. Může totiž nastat situace, kdy algoritmus vrátí množinu 500 různých, pseudonáhodně uspořádaných řešení. V takovém případě by výběr toho nejvhodnějšího mohl být pro člověka netriviální a časově náročnou úlohou.

Řazení optimálních řešení

Vhodnost řešení jsem se rozhodl vyhodnocovat na základě rovnoměrnosti rozdělení účastníků mezi rundami. Tento způsob nejlépe reflektuje požadavky z praxe, neboť umožňuje spojování disciplín v rámci jedné rundy, aniž by se měnily konflikty, a zároveň respektuje skutečnost, že se počet instruktorů na disciplíně (může jich být více) odvíjí od počtu účastníků na disciplíně. Rovnoměrné rozdělení účastníků umožňuje rovnoměrné rozdělení instruktorů, takže nebudou vznikat situace, kdy je v jedné rundě nedostatek instruktorů a v další rundě nemají instruktoři co na práci. Řešení s rovnoměrným rozdělením účastníků mezi rundami jsou tedy rozhodně vhodnější, než řešení s nerovnoměrným rozdělením účastníků.

Rovnoměrnost rozdělení účastníků mezi rundami je možné spočítat pomocí rozptylu dle následujícího vzorce:

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - E(x))^2$$

kde n je počet rund (n=3), x_i je počet účastníků v rundě i a $E(x)$ je střední hodnota počtu účastníků v rundě. Vzhledem k tomu, že není nutné znát přesnou hodnotu rozptylu, ale stačí porovnat hodnoty rozptylů dvou řešení, je možné tento vzorec zjednodušit díky následující matematické vlastnosti. Pokud vynásobíme dvě čísla stejným kladným číslem, nezmění se poměr těchto čísel (a tedy ani jejich pořadí na lineární ose). Rozptyly účastníků na rundách dvou řešení tedy lze vynásobit stejným číslem, přičemž bude možné takto získanou hodnotu použít pro měření a srovnávání vhodností různých řešení. Po vynásobení rozptylu hodnotou n vznikne vzorec pro výpočet veličiny, kterou jsem pro účel této práce pojmenoval *disperze*.

$$disperze = \sum_{i=1}^{n}(x_i - E(x))^2$$

Použití disperze namísto rozptylu má jednu výhodu - při výpočtu vyžaduje o jednu operaci dělení méně, což by při počítání této hodnoty pro desítky tisíc řešení mohlo ulehčit práci zařízení provádějícímu výpočet.

Změna volby účastníka v konfliktu

Po běhu algoritmu provádějícího hledání množiny řešení s minimálním počtem konfliktů může nastat situace, kdy nalezená řešení mají nenulový počet konfliktů. V takovém případě je nutné, aby účastníci v konfliktu změnili jednu ze svých voleb. Konflikt je totiž situace, kdy má jeden účastník obě své vybrané disciplíny ve stejné rundě. K vyřešení konfliktu tedy stačí, aby jedna z těchto voleb disciplín byla změněna na disciplínu z jiné rundy. Systém tedy musí po výběru nejvhodnějšího z nalezených řešení také umožnit instruktorovi provést změnu volby účastníka v konfliktu. Touto změnou se pouze snižuje počet konfliktů, samotné rozdělení disciplín do rund zůstává nezměněno, takže nebude nutné znovu spouštět algoritmus hledající řešení s minimálními počty konfliktů.

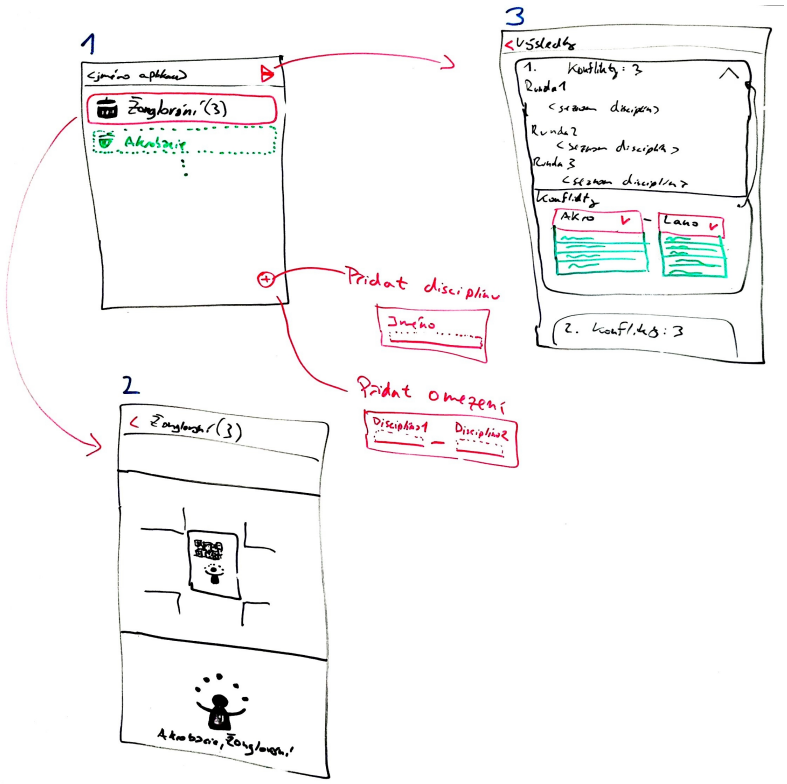
4.3 Návrh uživatelského rozhraní

Na základě návrhu jednotlivých komponent systému pro řešení problému bylo možné vytvořit návrh uživatelského rozhraní, ze kterého budu vycházet při implementaci. Téměř všechny komponenty je možné implementovat na mobilní platformě, výjimkou může být výpočetní

algoritmus hledající množinu řešení s minimálním počtem konfliktů. Je totiž možné, že výpočet bude pro mobilní zařízení příliš zdlouhavý. Pro výpočet však není nutné vytvářet uživatelské rozhraní a v případě nutnosti by jej bylo možné provádět na serveru, který by svou odpověď na požadavek výpočtu odesílal zpět mobilní aplikaci, takže nic nebrání tvorbě návrhu uživatelského rozhraní.

Aplikace se bude skládat ze tří hlavních obrazovek, každá bude mít svou unikátní funkci. Pro zajištění intuitivního ovládání se návrh odvíjí od grafického stylu Material Design, jenž se snaží propojit jednoduchost s praktičností a je v současné době velmi používaný ve světě (nejen) mobilních aplikací.

Při návrhu aplikace jsem pracoval s bílou tabulí, která mi umožňovala provádět jeho změny během průběžného uživatelského testování. Snímek této tabule společně s konečným návrhem je přítomen na obrázku 4.2.



Obrázek 4.2: Návrh uživatelského rozhraní. Obrazovka 1 je domovská, obrazovka 2 je určena pro přidávání účastníků do disciplín a obrazovka 3 prezentuje výsledky výpočtu spolu s možností řešení konfliktů. Červeně jsou naznačeny interaktivní prvky sloužící k zásadní změně obrazovky, zeleně vyznačen prostor na další potenciální prvky stejného typu. Návrh byl vytvořen na bílé tabuli.

Domovská obrazovka

Na domovské obrazovce je možné přidat jednotlivé disciplíny a omezení pomocí plovoucího tlačítka v pravém dolním rohu. Tlačítko zpřístupňuje formuláře umožňující zadat název vytvářené disciplíny, nebo vybrat dvě disciplíny tvořící omezení. Po potvrzení se dlaždice příslušné disciplíny, nebo omezení objeví v těle obrazovky. K disciplínám je možné přiřadit účastníky klepnutím na příslušnou dlaždici disciplíny, tato akce zpřístupní uživateli

obrazovku pro přidávání účastníků (bude popsána níže). Počet účastníků v disciplíně je zobrazen na dlaždici hned vedle jejího názvu. Každá dlaždice také obsahuje ikonu odpadkového koše umožňující smazat příslušnou disciplínu, nebo omezení. V případě disciplín jsou při stisknutí ikony odstraněni její účastníci. Pokud však disciplína nemá přiřazeného žádného účastníka, je odstraněna samotná disciplína. Toto umožňuje odstranit pouze účastníky a zachovat disciplínu. V pravém horním rohu se také nachází tlačítko spouštějící výpočet algoritmu rozdělení a následné přenesení uživatele na obrazovku výsledků výpočtu.

Obrazovka pro přidávání účastníků do disciplín

Na aplikační liště obrazovky je uveden název disciplíny, do které jsou aktuálně přidáváni účastníci a jejich počet. Nechybí ani tlačítko pro vrácení se na domovskou obrazovku. Tělo obrazovky tvoří výřez pro kameru, skrze kterou budou skenovány karty účastníků (popsány v sekci 4.1). Pod výřezem se nachází ikona posledního skenovaného účastníka a jeho volby, aby bylo předejito případným chybám při načítání. Účastník bude k disciplíně přiřazen, pouze pokud má zapsánu jednu nebo žádnou disciplínu. Je tedy možné využít tuto obrazovku ke kontrole disciplín účastníka, který má již naskenované obě volby.

Obrazovka výsledků výpočtu

Zde se zobrazí všechna nalezená řešení, každé na své rozbalitelné dlaždici. V kompaktním modu dlaždice je zde vidět počet konfliktů, jednotlivé rundy a jim příslušející disciplíny. Po rozbalení dlaždice se objeví případné konflikty a tlačítka umožňující jejich řešení. Návrh této obrazovky je velmi hrubý, pravděpodobně bude pozměněn během implementace na základě průběžného uživatelského testování.

Kapitola 5

Implementace mobilní aplikace

Tato kapitola se zabývá zajímavými částmi implementace systému pro podporu řešení problému optimálního rozdělení disciplín a kromě toho poskytuje všeobecné informace o vytvořené aplikaci. Před samotnou implementací byla vytvořena hrubá podoba algoritmu Branch and Bound (popsaného v sekci 3.1) pro zjištění, zda mobilní zařízení poskytuje dostatečný výkon pro nalezení optimálního rozdělení disciplín. Tento test proběhl úspěšně, a tak jsem nemusel přistupovat ke krokům, jakým je například provádění výpočtu na odlišné, výkonnější platformě.

5.1 Použité technologie

Pro implementaci aplikace byl použit vývojový kit Flutter používající programovací jazyk Dart. Aplikace vytvořené použitím tohoto kitu umožňují AOT[1] kompilaci do nativního kódu příslušného procesoru (díky čemuž je zajištěna kompatibilita pro Android a iOS). Také umožňuje JIT[2] kompilaci byte kódu pro interpretaci vlastním virtuálním strojem, což je užitečné zejména při vývoji aplikací díky možnosti tzv. Hot reload[3]. Flutter je poměrně novým nástrojem, první stabilní verze vyšla 12. dubna 2018, proto nemusí být některé zásadní elementy pro tvorbu aplikací implementovány přímo v jazyce, je ale možné využít balíčků vydávaných třetími stranami [8].

Flutter byl vybrán, jelikož v něm vytvořené aplikace pracují výrazně rychleji než při použití frameworků jako React Native [5], ale zároveň je možné vytvořenou aplikaci spouštět na obou nejpoužívanějších mobilních operačních systémech Android a iOS (na rozdíl od jazyků Java, Kotlin či Swift), což by mohlo být užitečné v případné budoucí práci. Pokud by se ukázalo, že použití Flutteru na mobilní platformě neposkytuje dostatečný výkon či paměť pro řešení, uvažoval bych o zapojení webového serveru do celkového systému pro podporu řešení problému optimálního rozdělení disciplín. Mobilní aplikace by pak zasílala HTTP požadavek obsahující vstupní data a server by po výpočtu odeslal příslušnou odpověď.

Některé konstrukce jazyka Dart

Dart vychází z jazyka C (syntaxe je velmi podobná), ale je vysokoúrovňový, tudíž obsahuje jisté odlišnosti. Pro lepší orientaci ve zdrojovém kódu aplikace tedy uvádím vybrané zá-

[1] Ahead-of-time, kompilace proběhne před spuštěním aplikace

[2] Just-in-time, kompilace probíhá při běhu aplikace

[3] Modifikace zdrojového kódu jsou uplatněny za běhu aplikace, aniž by muselo dojít k jejímu restartu a je zachován její stav.

sadní konstrukce a specifika tohoto jazyka. Je předpokládána znalost základních principů objektově orientovaných jazyků.

- Datový typ `List<Class>` znázorňuje seznam instancí třídy `Class` libovolné délky (dokud je k dispozici paměť). Je možné ho využívat různými způsoby, například jako pole (kdy se přistupuje prvkům pomocí indexu `myList[2]`, či `myList.elementAt(2)`), či jako zásobník (kdy se pro vkládání a vyjímání prvků využívají instanční metody `add(Class item)` a `removeLast()`). Je možné tvořit i seznamy základních datových typů (například `List<int>`), neboť i ty jsou v jazyce Dart třídami.
- Dart je silně typovaný, je však možné použití typu `dynamic`, který umožňuje použití libovolného datového typu.
- Identifikátor začínající podtržítkem, ať jde o atribut, či metodu, značí jeho soukromou viditelnost (ekvivalent `private` v např. jazyce Java). Naopak absence podtržítka jakožto prvního znaku identifikátoru značí veřejnou viditelnost (ekvivalent `public`).
- Třídy dle konvence začínají velkým písmenem.
- Statické třídy lze implementovat označením všech metod a atributů třídy klíčovým slovem `static`. Dart totiž neumožňuje označit celou třídu jako statickou. Pojmem "statický"je myšlena možnost přistupovat k metodě/atributu bez použití instance této třídy. Pro zavolání statické metody `method()` třídy `Class` by byl použit příkaz `Class.method()`.

Pomocí Flutteru je možné vyvíjet aplikace s použitím emulátoru, nebo reálného mobilního zařízení připojeného pomocí USB kabelu. Já jsem se rozhodl pro druhou volbu, jelikož umožňuje lepší vcítění se do koncového uživatele.

5.2 Implementace algoritmu pro hledání optimálních řešení problému

Algoritmus byl implementován na základě jeho návrhu v kapitole 3. Je implementován pomocí třídy `Solver`. Tato třída využívá dalších tříd reprezentujících řešení, disciplíny či účastníky, díky kterým je možné realizovat výpočet optimálních rozdělení disciplín. V této sekci jsou klíčové prvky zmíněných tříd detailně rozebrány.

Třída `Solver`

Tato třída obsahuje následující atributy (jsou uvedeny pouze atributy významné pro výpočet):

`static List<Solution> solutions` - seznam dosud nalezených optimálních řešení (listů stromu stavového prostoru). Po skončení běhu algoritmu bude obsahovat globálně optimální řešení.

`static List<Solution> _solutionStack` - zásobník rozpracovaných řešení (uzlů stromu stavového prostoru). Jedná se o privátní atribut (jeho název začíná podtržítkem).

`static int minConflicts` - dosavadní hodnota minimálního počtu konfliktů.

Kromě výše zmíněných atributů obsahuje třída Solver i jedinou veřejnou metodu `solve()`, která implementuje celý proces hledání optimálního rozdělení. Přitom využívá zbývajících privátních metod této třídy. Pracuje se zásobníkem řešení, na kterém je zpočátku uloženo jedno prázdné (s prázdnými rundami) řešení. Metoda provádí iteraci: vyjme řešení z vrcholu zásobníku a zpracuje ho metodou _branchAndBound(Solution), která případná nekoncová řešení vkládá zpět na zásobník. Tato iterace probíhá do doby, než je zásobník prázdný.

```
static solve() {
  _init();
  Solution s;
  while( _solutionStack.isNotEmpty ){
    s~= _solutionStack.removeLast();
    _branchAndBound(s);
  }
  _sortSolutions();
}
```

Metoda `_init()` zajišťuje inicializaci celého hledání optimálních rozdělení disciplín. Inicializuje atribut `solutions` na prázdný seznam, atribut `_solutionStack` na seznam s jediným prázdným řešením a nastaví počítadlo `minConflicts` na hodnotu počtu účastníků (konfliktů bude i v nejhorším případě méně, než je počet účastníků).

Metoda `_sortSolutions()` řadí po výpočtu rozdělení všechna nalezená řešení podle tzv. disperze (viz sekce 4.2).

Metoda `_branchAndBound(Solution)` zajišťuje větvení a případné zahazování řešení. Detailně bude metoda popsána dále v této sekci, neboť je nutné pro pochopení jejího fungování vědět více o třídách `Solution`, `Discipline` a `Person`.

Třída `Solution`

Každá instance třídy `Solution` znázorňuje jeden uzel stromu stavového prostoru. Nemusí tedy jít nutně o koncové řešení (list). Třída obsahuje následující atributy:

- 3 rundy implementované pomocí 3 seznamů typu `List<Discipline>`
- Počítadlo `int _indexToBeInserted` uchovávající index disciplíny, která má být jako další do daného nekompletního (nelistového) řešení vložena. Předpokládá se existence nějakého seznamu disciplín, stejného v rámci celého výpočtu.
- Seznam účastníků `List<Person> conflictingParticipants`, kteří jsou pro danou instanci řešení v konfliktu.
- Takzvanou disperzi (`double dispersion`), která se používá pro porovnání vhodnosti dvou disciplín se stejným počtem konfliktů.

Pro získání počtu konfliktů dané instance řešení `s` stačí použít příkaz `s.conflictingParticipants.length`, neboť účastník je v konfliktu vždy sám se sebou. Bylo by jistě možné mít pro tuto hodnotu samostatný atribut, díky kterému by bylo potenciálně rychlejší získání počtu konfliktů, v dokumentaci jazyka Dart se však tento postup nedoporučuje. Dart totiž udržuje informaci o délce seznamu ve své vnitřní paměti a ukládání této hodnoty by bylo redundantní.

Třída `Discipline`

Třída obsahuje dva statické atributy: atribut `disciplines` obsahuje seznam všech disciplín (využívaný společně s indexem uloženým v instanci `Solution` k určení vkládané disciplíny při větvení) a atribut `allLimitations` obsahující seznam všech omezení. Omezení je implentováno jako dvojice disciplín (opět pomocí seznamu).

Kromě toho náleží každé instanci atributy:

- `String name` obsahující název disciplíny (pomocí kterého je identifikována, nemohou existovat dvě disciplíny se stejným názvem).
- `List<Person> participants` obsahující seznam účastníků, kteří by rádi trénovali tuto disciplínu.
- `List<Discipline> limitations` obsahující seznam disciplín, s nimiž tato disciplína nesmí sdílet rundu kvůli existujícímu omezení.

Oba seznamy (`participants` a `limitations`) slouží k rychlejšímu zjišťování počtu konfliktů daného rozdělení. Vždy stačí zkoumat pouze to, zda runda, do které se vkládá, obsahuje disciplíny uvedené v `limitations`, případně disciplíny zapsané u účastníků `participants`. Výpočet je tedy urychlen za cenu větší paměťové náročnosti.

Třída `Person`

Třída obsahuje statický atribut `people` obsahující seznam všech vytvořených účastníků. Kromě toho náleží každé instanci atributy:

- `String name` obsahující jméno účastníka. Nejedná se o skutečné jméno typu `Jan Novák`, ale o kód dané karty účastníka, na kterou je účastník podepsán, tedy například `stojka_R`.
- `List<Discipline> choices` obsahující disciplíny, které si daný účastník zapsal. Seznam je aplikačně limitován na dva objekty, což koresponduje se dvěma volbami, které si každý účastník zapisuje.

Důvod těsné provázanosti tříd `Person` a `Discipline` byl uveden v popisu třídy `Discipline`.

Implementace metody `_branchAndBound(Solution parent)`

Metoda kombinuje horlivou a línou variantu algoritmu Branch and Bound (viz sekce 3.1). Nejprve ověří, zda počet konfliktů (délka seznamu `conflictingParticipants` daného řešení `parent`) není větší než minimální nalezený počet konfliktů v proměnné `minConflicts`. Pokud by byl větší, je možné toto řešení zahodit a skončit metodu (algoritmus by pak vyjmul další řešení ze zásobníku a pokračoval).

Následně je řešení rozvětveno. Větvení probíhá postupným klonováním rodičovského řešení a přiřazováním další disciplíny v řadě, každému klonu je do jedné (pokaždé jiné) rundy vložena tato disciplína. Pro zamezení výskytu identických řešení může být disciplína během větvení vložena jen jednou do prázdné rundy (viz 3.2). Při větvení jsou také kontrolována omezení. Pokud by měla být vložena do rundy disciplína, která je společně s jinou již přítomnou disciplínou součástí omezení, pak se toto vložení neuskuteční a vytvořený klon je zahozen.

Po vložení disciplíny je nutné přepočítat počet konfliktů. Přepočítání je provedeno metodou `_countConflicts` a spočívá v nalezení společného účastníka mezi vkládanou disciplínou `newDiscipline` v disciplínách rundy `round`. Pokud k tomuto nalezení dojde, je příslušný účastník přidán do seznamu účastníků v konfliktu `conflictingParticipants`. Vzhledem k tomu, že každý účastník má zapsané právě dvě disciplíny, je možné použít po nalezení účastníka příkaz `break`, který způsobí začátek zkoumání konfliktů dalších účastníků vkládané disciplíny.

```
void _countConflicts(List<Discipline> round, Discipline newDiscipline){
  for(Person newParticipant in newDiscipline.participants){
    for( Discipline oldDiscipline in round ){
      if(oldDiscipline.participants.contains(newParticipant)){
        this.conflictingParticipants.add(newParticipant);
        break;
      }
    }
  }
}
```

Metoda `branch()` instance `Solution` tedy může vracet seznam synovských řešení o délce 1-3. Tato řešení je třeba dále zpracovat, o což se stará následující část metody `_branchAndBound(Solution parent)`:

```
if(children[0].isFinal(nrOfDisciplines)){
  for(Solution child in children){
    /* bounding final */
    if(child.conflicts() < Solver.minConflicts){
      Solver.minConflicts = child.conflicts();
      Solver.solutions = [child];
    }
    else if(child.conflicts() == Solver.minConflicts){
      Solver.solutions.add(child);
    }
  }
}else{
  for(Solution child in children){
    if(child.conflicts() <= Solver.minConflicts){
      Solver._solutionStack.add(child);
    }
  }
}
```

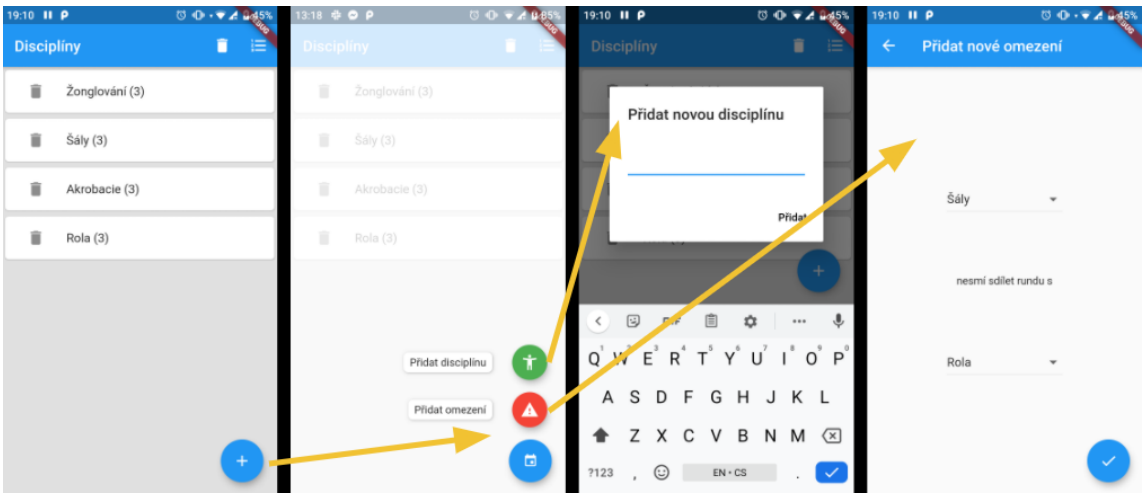
Nejprve je zjištěno, zda jsou vytvořená řešení konečná. O to se stará instanční metoda `isFinal()` třídy `Solution`, která porovná celkový počet disciplín uložený jako statický atribut třídy `Solver` s indexem dalšího vkládaného řešení uloženého v atributu `_indexToBeInserted`. Toto je možné, neboť jsou listové uzly uloženy vždy ve stejné úrovni a stejnou úroveň mají i řešení získaná větvením. Pokud je tedy první z řešení získaných větvením konečné (listové), pak jsou konečná i všechna ostatní. V případě konečných řešení se počet konfliktů každého takového řešení porovná s počtem dosud nejlepších řešení. Je-li

nové řešení lepší (má méně konfliktů) než nejlepší dosud nalezené, pak jsou všechna dosud nalezená řešení nahrazena nalezeným řešením a `minConflicts` je nastaven na novou hodnotu. Pokud má nové řešení stejný počet konfliktů, pak je pouze přidáno do seznamu zatím nejlepších řešení `solutions`. Ve zbytku případů je řešení zahozeno. V případě, kdy řešení získaná větvením nejsou konečná, jsou tato řešení přidána na zásobník rozpracovaných řešení `_solutionStack`, pokud jsou zatím alespoň stejně dobrá jako nejlepší dosud nalezená řešení.

5.3 Implementace uživatelského rozhraní

Grafická rozhraní se ve Flutteru tvoří pomocí vrstvení objektů třídy `Widget`. Existují dvě třídy rozšiřující tuto třídu, a to `StatefulWidget` (umožňující uchovávat a hlavně měnit stav vykresleného objektu) a `StatelessWidget` (po vykreslení není možné nijak modifikovat). Každá instance třídy `Widget` má konstrukční metodu `build`, která vrací objekt (může obsahovat další objekty), jenž má být vykreslen. Kromě toho je možné v objektech třídy `StatefulWidget` volat metodu `setState(voidCallback fn)`, která zajistí překreslení obrazovky. Parametrem je funkce, která by měla obsahovat změny dat, jež se mají po překreslení projevit.

Implementace domovské obrazovky



Obrázek 5.1: Domovská obrazovka a formuláře umožňující vkládat disciplínu či omezení. Žluté šipky značí přístup k formulářům, jedná se o snímky různých stavů obrazovky.

Na každé dlaždici (první snímek obrázku 5.1) disciplíny je ikona koše. Ta slouží pro mazání účastníků disciplíny, případně pro mazání disciplíny samotné (pokud nemá žádné účastníky). Při mazání je odstraněn záznam o účastníkovi v disciplíně a zároveň záznam o disciplíně ve volbách účastníka. Číslo vpravo od názvu disciplíny značí počet přihlášených účastníků. Pro vytvoření menu pro výběr přidávaného prvku (disciplíny či omezení) bylo použito balíčku `flutter_speed_dial` verze 1.2.5[4].

[4] https://pub.dev/packages/flutter_speed_dial

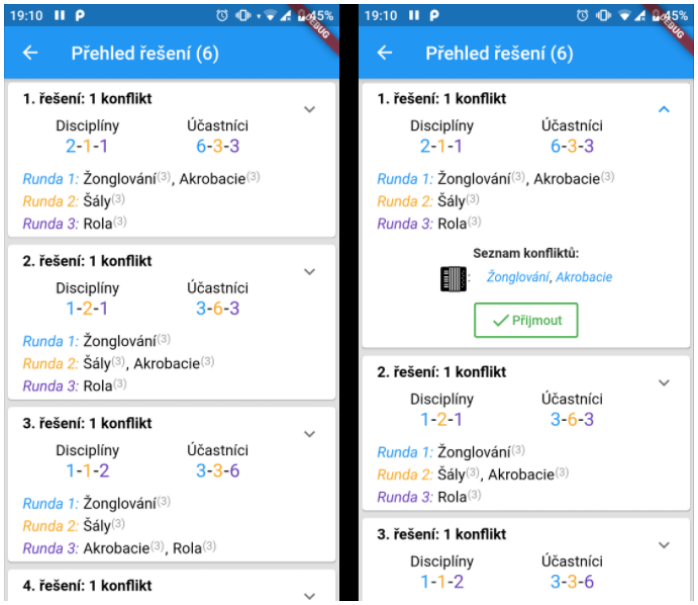
Implementace obrazovky disciplíny

Obrázek 5.2: Obrazovka disciplíny (v tomto případě Žonglování), na které je možné přiřazovat k disciplíně účastníky.

V horní části obrazovky na obrázku 5.2 je výřez pro kameru mobilního zařízení, která snímá karty účastníků pomocí balíčku `flutter_qr_bar_scanner` verze 1.0.1[5]. Každý naskenovaný snímek je vyhodnocen, a je-li v něm nalezen QR kód, je jeho obsah předán zpracovávající funkci. Tato funkce nalezne (případně vytvoří nového) účastníka, který (pokud má méně než dvě disciplíny) je přiřazen do aktuálně otevřené disciplíny (její název je v záhlaví). Kromě toho je zobrazena ikona odpovídající ikoně na kartě účastníka a účastníkovy dosud zapsané disciplíny, aby se předešlo nechtěnému naskenování. Na obrazovce je také vidět celkový počet účastníků, změna tohoto čísla signalizuje úspěšné naskenování karty. Po uživatelském testování jsem se ještě rozhodl signalizovat vložení účastníka do disciplíny krátkou vibrací.

[5] https://pub.dev/packages/flutter_qr_bar_scanner

Implementace obrazovky výsledků výpočtu

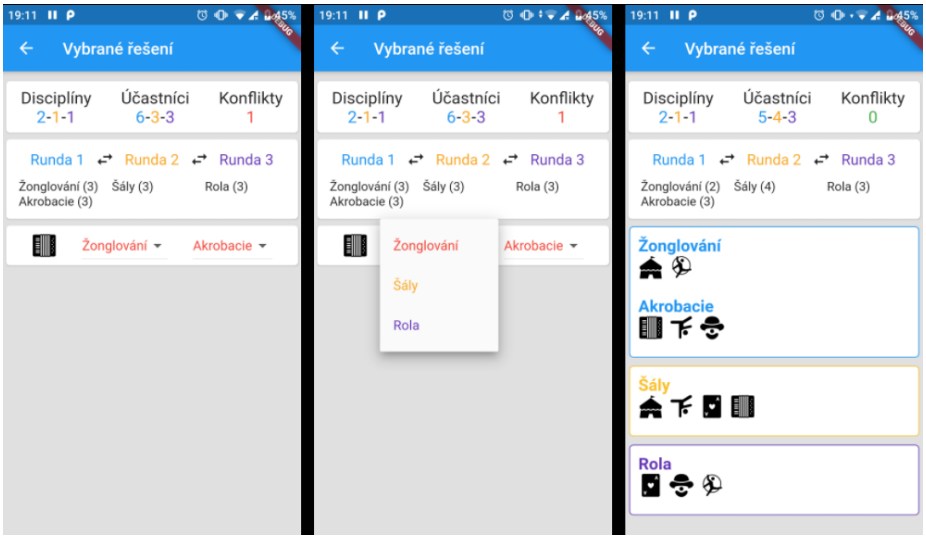


Obrázek 5.3: Obrazovka výsledků a jejich důležitých parametrů, každý výsledek je možné expandovat. Díky tomu se zobrazí případní účastníci v konfliktu a tlačítko umožňující daný výsledek vybrat. Na obrázku jsou zobrazeny snímky s neexpandovanou a expandovanou dlaždicí disciplíny (respektive).

Výsledky jsou řazeny pomocí tzv. disperze popsané v sekci 4.2. Dále jsou pro výběr vhodného rozdělení zobrazeny (viz obrázek 5.3) důležité informace, jako je počet a seznam disciplín či počet účastníků v jednotlivých rundách. Konkrétní účastníci v konfliktu jsou v základním pohledu skryti, neboť by mělo být jedno, kteří účastníci budou měnit svá rozhodnutí (instruktor vybírající rozdělení by neměl být ovlivněn identitou účastníků v konfliktu). Je možné však každé řešení expandovat a symboly spolu s vybranými disciplínami těchto účastníků zjistit. Pro implementaci je použit widget `ListView`, který se stará o vykreslování aktuálně zobrazených dlaždic. Není nutné počítat obsah a rozložení dlaždic mimo obrazovku a tento způsob šetří zdroje mobilního zařízení.

Implementace obrazovky pro řešení konfliktů

Tato obrazovka byla oproti návrhu přidána na základě průběžného uživatelského testování, jelikož bylo řešení konfliktů na obrazovce s ostatními výsledky výpočtu nepřehledné.



Obrázek 5.4: Obrazovka pro řešení konfliktů. Pokud existují v daném řešení konflikty, jsou vypsány v seznamu pod výpisem disciplín v rundách (viz 1. snímek). Na druhém snímku je zobrazen výběr vhodné disciplíny namísto Žonglování, které je v konfliktu s Akrobacií. Po vybrání disciplíny z jiné rundy se obrazovka přepne do podoby na třetím snímku a je vidět i finální rozdělení účastníků do disciplín.

Kromě seznamu disciplín v rundách je u každé disciplíny vidět (obrázek 5.4) i počet jejich účastníků. Ten by mohl být nápomocný při výběru nové disciplíny účastníka v konfliktu. Vybírání nové disciplíny je zjednodušeno i pomocí barev, disciplíny v selektoru mají barvy rundy, do které náleží. Po výběru jedné nové disciplíny je konflikt vyřešen a z přehledu zmizí. Po odstranění všech konfliktů se zobrazí finální rozdělení účastníků v disciplínách. Rundy je možné mezi sebou libovolně prohazovat pomocí šipek mezi nimi, na konflikty toto nemá žádný vliv.

Kapitola 6

Testování vytvořeného systému pro řešení problému rozdělení

V rámci testování vytvořené aplikace jsem se, kromě funkčního testování aplikace, soustředil na dva zásadní aspekty: uživatelské testování a analýzu škálovatelnosti. Během uživatelského testování jsem se zaměřil na to, jak je vytvořená aplikace vnímána koncovým uživatelem a zda mu opravdu zjednoduší práci. Analýzu škálovatelnosti jsem vypracoval, abych mohl určit limity vytvořené aplikace a dala se stanovit hranice množství dat, s jakou je aplikace stále použitelná. Oba tyto aspekty podrobně rozeberu v této kapitole a ze získaných informací budu moci čerpat v úvahách o budoucí práci popsaných v kapitole 7.

6.1 Uživatelské testování

Uživatelské testování proběhlo vzhledem k probíhající pandemii koronaviru v omezené míře, a nejsou v něm z tohoto důvodu použita reálná data. Jsou však použita data ve stejném rozsahu a komplexitě a uživatelské testování by tak mělo být jen minimálně ovlivněno. K testování se uvolili dva trenéři, kterým byla aplikace nejprve představena. Následně byli postaveni před úkol najít vhodné rozdělení disciplín do rund na základě předem stanoveného, víceméně náhodného rozdělení dětí do disciplín ve formě různě velkých kupiček karet účastníků. Každá kupka byla označena štítkem s názvem disciplíny, kterou reprezentuje. Celkem bylo rozděleno 60 účastníků (tedy 120 karet) do 16 disciplín se třemi disciplínami, které vzájemně nemohly sdílet rundu (tedy 3 omezení).

Průběh testů

Každý z testerů vždy zadal název disciplíny a následně do ní naskenoval veškeré její účastníky. Následně zadal omezení a spustil výpočet. Nakonec vybral dle svého úsudku nejvhodnější řešení (pracovalo se s fiktivními daty, takže nebylo možné kalkulovat s dostupností jednotlivých instruktorů specializovaných v konkrétních disciplínách) a vyřešil konflikty na základě mých pokynů (nebyli bohužel k dispozici reální účastníci, kteří by se rozhodovali). Nakonec každý z testerů zkompletoval karty účastníků po dvojicích, aby byl celý systém připraven pro další použití.

Výsledky měření

Každý z obou testů byl měřen s následujícími výsledky (prezentovány ve formátu minuty:vteřiny, měřeno pomocí stopek):

Aktivita	Čas prvního testu	Čas druhého testu
Zapisování disciplín a účastníků	8:27	7:01
Vkládání omezení	1:14	1:02
Kontrola počtu účastníků v disciplínách	1:47	2:42
Výpočet aplikace	0:00	0:00
Kompletace karet účastníků	6:31	6:46
Celkové časy	17:59	17:31

Významný rozdíl v časech zápisu disciplín způsobilo nechtěné smazání účastníků disciplíny jedním z testerů. V obou případech byl výpočet téměř instantní. Celý proces by mohl být urychlen delegováním kontroly počtu účastníků na jiné instruktory (sečtou účastníky na disciplíně a jen si s uživatelem aplikace ověří počet). Kromě toho by bylo možné podobným způsobem delegovat kompletaci karet účastníků, případně na tento proces využít účastníky a provádět ho formou nějaké hry. Po těchto urychleních bude dle mých odhadů možné celý proces realizovat během 10-15 minut s garantovaným nalezením optimálního řešení, což je oproti původním 40 minutám a žádnou garancí optimálnosti výrazné zlepšení.

Zpětná vazba

Kromě drobné stížnosti, že laminované karty účastníků měly ostré rohy (což bylo obratem vyřešeno jejich zaoblením), si jeden z testerů stěžoval na fakt, že ukončením aplikace jsou smazána veškerá data. Po poradách s jednotlivými testery jsme však přišli na skutečnost, že zadaná data vlastně není potřeba uchovávat, jelikož aplikaci je nutné pro jednu várku vstupních dat spustit jen jednou.

Při srovnávání nalezených řešení měl jeden z testerů poznámku, že dlaždice s detaily o řešení obsahovala příliš mnoho informací a působila přeplněně. Toto byl dle slov testera však jen prvotní pohled, všechny informace se totiž hodily k výběru optimálního řešení (i například z hlediska potenciálního spojování disciplín).

Další připomínkou byla skutečnost, že při skenování účastníků musí instruktor bedlivě sledovat symbol naposledy skenovaného účastníka, aby si byl jistý, že příslušného účastníka opravdu naskenoval. Toto sledování si žádá neustálou pozornost instruktora, což není uživatelsky zcela přívětivé. Proces by mohl být zjednodušen, kdyby byl instruktor nějak notifikován v případě úspěšného naskenování. Proto jsem na základě této připomínky upravil aplikaci, aby po úspěšném naskenování způsobila vibraci mobilního zařízení po dobu 60 milisekund.

Mimo tyto připomínky sklidil systém kladné ohlasy, zejména byla pozitivně hodnocena rychlost a jednoduchost zadávání vstupních dat spolu s rychlostí výpočtu (což byly u obou testerů primární obavy).

6.2 Analýza škálovatelnosti

Při testování limitů systému jsem se zaměřil pouze na schopnost algoritmu hledajícího optimální rozdělení disciplín pracovat s více disciplínami a účastníky, neboť se jedná o část nejnáchylnější k problémům při takovémto škálování (kartiček je jednoduchým způsobem

možné vytvořit více a vykreslovací prvky aplikace nejsou početně limitovány). Před zahájením výpočtu jsem tedy využil novou funkci `solverTestCase(int participantCnt, int disciplineCnt)`, která vytvoří poskytnutý počet disciplín a účastníků, jež jsou mezi disciplíny (pseudo)náhodně rozděleni. Pro měření uplynutého času mezi začátkem a koncem běhu algoritmu pro výpočet optimálního rozdělení používám třídy `Stopwatch`, která je součástí programovacího jazyka Dart.

Počet disciplín	Počet účastníků	Průměrný čas
40	16	<0,5 s
80	16	2,5 s
100	16	5 s
150	16	16 s
40	20	0,5 s
80	20	17,5 s
100	20	36 s
150	20	157,5 s
40	25	58,5 s
80	25	56 s
100	25	163 s
150	25	389 s
160	32	>1 hodina

Měření proběhlo v pěti opakováních pro každý záznam v tabulce, uvedený čas je průměrem těchto hodnot zaokrouhlený na poloviny sekund (pro ilustraci doby výpočtu bohatě postačuje). Výjimkou je poslední test se 160 účastníky a 32 disciplínami, který jsem spouštěl jen dvakrát. Každé z opakování trvalo více než hodinu a použití systému pro takovéto parametry se mi jeví jako nevhodné.

Při 25 disciplínách a 40 účastnících začala zařízení při některých opakováních docházet paměť z důvodu příliš mnoha nalezených řešení s nulovým počtem konfliktů a aplikace byla v těchto případech terminována. Z 15 pokusů takováto situace nastala jen dvakrát, i tak to ale signalizuje limit mobilního zařízení. Upravil jsem tedy zdrojový kód, který v případě 400 nalezených řešení s nulovým počtem konfliktů tato řešení vrátí (400 řešení k výběru je více než dost) a v případě nalezených více než 500 řešení s nenulovým počtem konfliktů jich ponechá 100 (ostatní náhodně zahodí). Tento způsob řešení není nejvhodnější, jelikož přestává (v případě zahazování řešení s nenulovým počtem konfliktů) garantovat optimálnost nalezených řešení. Pro účel nalezení limitu škálovatelnosti však postačuje.

Z testování lze vyvodit, že systém není vhodné používat s více než 25 disciplínami, pokud je třeba garantovat optimálnost řešení. Při použití s více disciplínami a účastníky je nutné počítat se zvýšenou dobou výpočtu, která při 32 disciplínách a 160 účastnících přestává být únosná. Rozhodně však mohu prohlásit, že systém pracuje skvěle v podmínkách, pro které je určen.

Kapitola 7

Úvahy o možné budoucí práci a závěrečné shrnutí

V předchozích kapitolách jsem se zaobíral tím, jak vytvořit systém řešící problém hledání optimálního rozdělení disciplín, který v současnosti spočívá v nalezení rozdělení pro orientačně 16 disciplín a 60 účastníků. Je však vhodné zaobírat se případem, kdy by systém přestal postačovat, protože možnost snadné rozšiřitelnosti jakéhokoliv systému je z hlediska jeho dlouhodobé udržitelnosti klíčová. V této kapitole se budu zabývat možnostmi, které se s aktuálně dostupnými technologiemi nabízí. Také se zde nachází shrnutí celé bakalářské práce.

7.1 Možné úpravy pro zlepšení systému

V současnosti zvládá celý sytém fungovat na mobilním zařízení, bez nutnosti serveru či připojení k internetu. Také zajišťuje nalezení řešení s globálně minimálním počtem konfliktů. Má však své limitace, co se týče objemu zpracovávaných dat. Pokud by bylo nutné kapacitu systému zvýšit, je pravděpodobné, že by bylo třeba obětovat některé aspekty (například právě fungování čistě na mobilním zařízení). Podobnými změnami za účelem zvýšení kapacity systému se zabývá tato sekce. Návrhy zde uvedené nebyly ke dni odevzdání bakalářské práce realizovány, jedná se o jakousi sumarizaci možností případné budoucí práce.

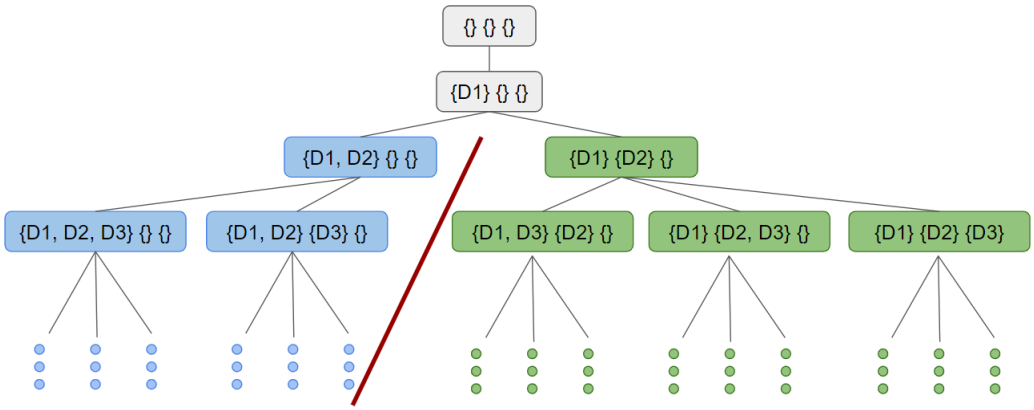
Zřízení serveru realizujícího výpočet

Mobilní zařízení jsou limitována svým výkonem a pamětí, nejsou stavěna pro řešení složitých výpočtů. Pokud by byl výpočet realizován na výkonnějším zařízení, posunula by se kapacita disciplín nebo účastníků, které by systém zvládl zpracovat. Nastal by však problém, jelikož by bylo ohroženo efektivní vkládání uživatelských dat. Pro jeho zachování by byla úloha výkonnějšího zařízení pouze ve výpočtu a pracovalo by jako server odpovídající na dotazy klienta ve formě mobilního zařízení.

Úloha mobilní aplikace by se zmenšila na zadávání vstupních dat a prezentaci výsledků. S výpočetním serverem by komunikovala pomocí předem definovaného protokolu. Nevýhodou tohoto řešení by byla nutnost buď připojení k internetu, nebo alespoň komunikace s výpočetním serverem. Věřím však, že pokud by byl systém využíván pro akce s takovými počty disciplín a účastníků, kdy by bylo třeba využívat serveru, nebylo by připojení k internetu problémem.

Distribuce výpočtu

Vzhledem ke způsobu řešení problému metodou Branch and Bound lze výpočet jednoduchým způsobem distribuovat mezi vícero zařízení, a tím pádem i urychlit, jak je demonstrováno na obrázku níže.



Obrázek 7.1: Demonstrace distribuce výpočtu. Červená dělicí čára znázorňuje hranici mezi modrou a zelenou částí, každá z těchto částí může být zpracována jiným zařízením. Takto lze problém rozdělit na libovolný počet podproblémů (obvykle dle počtu dostupných zařízení) v libovolné hloubce, každý z podproblémů je možné řešit individuálně a vzájemně paralelně.

Každému zařízení stačí pouze vědět, jaká je jeho počáteční úloha, jaké disciplíny se mají vkládat a jaké disciplíny jsou dosud v konfliktu. Pro zařízení zpracovávající modrou část (viz obrázek 7.1) by to byla počáteční úloha `{D1, D2} {} {}` a množina disciplín mimo již vložené `D1` a `D2`. Výsledkem každého výpočetního zařízení bude množina řešení společně s množinou disciplín v konfliktu. Tento výsledek je třeba předat zpět centrálnímu zařízení (které úlohu rozdělilo), jehož úkolem bude zjistit minimální počet disciplín v konfliktu mezi všemi výsledky a zahodit ty výsledky, které minimální počet konfliktních disciplín převyšují. Pokud by byl problém rozložen na velké množství podproblémů, mohla by jednotlivá zařízení odpovědná za výpočet postupně zpracovávat i vícero podproblémů. Vše by záleželo na konkrétní implementaci distribuce.

Výše zmíněný způsob řešení přináší jeden problém, a to že práce jednoho zařízení může být naprosto zbytečná. Může totiž nastat situace (odkazuji na obrázek 7.1), kdy lokální minima jednotlivých podstromů jsou rozdílná. Jedno ze zařízení začne zahazovat řešení s počtem konfliktů větším než tři, zatímco druhé akceptuje všechna řešení s počtem konfliktů menším než pět. Zařízení však mezi sebou nemohou nijak komunikovat, a tak dochází k plýtvání zdrojů i času druhého zařízení. Řešením by mohlo být sdílení hodnoty počtu konfliktů napříč zařízeními, například skrz centrální zařízení distribuující výpočet. V případě nalezení lepšího řešení by zařízení zaslalo zprávu centrálnímu zařízení, které by tuto zprávu rozšířilo mezi všechna ostatní zařízení. Otázkou je, jak časově náročný by byl proces komunikace a zda by se z časového hlediska vyplatil.

Zrychlení zadávání disciplín a účastníků

Pokud by se počet účastníků zvýšil do hodnot, kdy je v jednom člověku náročné naskenovat karty všech účastníků, bylo by nutné zvážit změnu způsobu zadávání účastníků, potažmo disciplín.

Zadávání disciplín by bylo možné urychlit vytvořením karet disciplín, kdy by do skenovaného kódu byla vložena informace, že se jedná o disciplínu s konkrétním názvem. Díky tomu by se opustilo zadávání názvů disciplín v textové formě a pro veškeré úkony spojené se zadáváním by sloužila čtečka.

K urychlení zadávání účastníků existují dva přístupy, žádný není v rozporu s navrženými zlepšeními v předchozích sekcích (dokonce by ta zlepšení byla pro implementaci urychlení zadávání užitečná).

- První přístup spočívá ve využití více čteček, které by naskenovaná data zasílaly na jedno centrální výpočetní zařízení. Na centrálním zařízení by byla data kompletována a spouštěl by se odsud (nebo nepřímo skrze mobilní zařízení) výpočet. Každá z čteček by pak musela naskenovat pouze část (namísto všech) disciplín a urychlilo by se zadávání účastníků.

- Druhou možností by byla úprava karet a čteček pomocí technologie RFID. Tato technologie umožňuje skenování tzv. RFID tagů pomocí elektromagnetického pole a nespoléhá na nutnost viditelnosti (jak je tomu například u QR kódů), a tak je možné skenovat všechny tagy v blízkém okolí čtečky takřka současně. Některé obchodní řetězce tuto technologii využívají k urychlení odbavení zákazníků u pokladen, kdy jsou všechny produkty vložené do pokladního koše téměř okamžitě načteny. Problémem tohoto řešení by byla vyšší pořizovací cena karet a případně čteček (mobilní zařízení vybavené technologií NFC by mělo být možné použít k účelu čtečky).

7.2 Závěr

V počátku této práce se podařilo důkladně analyzovat problém optimálního rozdělení disciplín. Následně byl navrhnut a implementován vhodný systém k řešení tohoto problému. Vznikla aplikace umožňující nejen velmi efektivní zadávání vstupních dat pomocí vytvořených karet účastníků (využívající technologie QR kódů), ale i realizující samotné hledání globálně optimálního řešení. Aplikace nevyžaduje připojení k internetu ani k žádnému výpočetnímu serveru, vše je hostováno lokálně na mobilní platformě. Bylo však pomýšleno i na případné škálování a kromě samotného systému řešící současnou podobu problému bylo i navrženo řešení umožňující pracovat s komplexnějšími podobami problému, na které by nebylo použití pouze mobilního zařízení dostatečné.

Uživatelské testování potvrdilo, že vytvořený systém pracuje nad očekávání, a je možné jej zapojit do provozu. Ve srovnání s dosud používaným řešením zabírajícím v průměru 40 minut je proces urychlen na 10-15 minut a namísto skupiny koordinovaných lidí vyžaduje jen jednoho člověka a mobilní telefon. Kromě vytvořené aplikace, dostupné na Google Play (`https://play.google.com/store/apps/details?id=com.ondrejholub.disciplinovac`), bylo vyrobeno 144 karet pro až 72 účastníků, které budou předány volnočasovému Cirkusu LeGrando. Karty účastníků je však možné v případě nutnosti dodělat, aniž by bylo nutné modifikovat aplikaci. Celkově považuji práci za vydařenou a těším se, až bude použita k účelu, pro který byla vytvořena.

Literatura

[1] Aickelin, U. a Dowsland, K. A. An indirect Genetic Algorithm for a nurse-scheduling problem. *Computers and Operations Research*. 31. vyd. 2004, č. 5, s. 761 – 778. Dostupné z: https://doi.org/10.1016/S0305-0548(03)00034-0. ISSN 0305-0548.

[2] Beagley, J. E. a Morris, W. *Clique Number and Chromatic Number of Graphs defined by Convex Geometries*. 2012. Dostupné z: https://math.nist.gov/mcsd/Seminars/2012/2012-10-18-Morris-presentation.pdf.

[3] Callahan, D. a Koblenz, B. Register Allocation via Hierarchical Graph Coloring. *SIGPLAN Not.* New York, NY, USA: Association for Computing Machinery. 26. vyd. 1991, č. 6, s. 192–203. Dostupné z: https://doi.org/10.1145/113446.113462. ISSN 0362-1340.

[4] Clausen, J. *Branch and Bound Algorithms - Principles and Examples*. 1999.

[5] Demedyuk, I. a Tsybulskyi, N. *Flutter vs Native vs React-Native: Examining performance* [online]. 2020. 2020 [cit. 14. 5. 2020]. Dostupné z: https://medium.com/swlh/flutter-vs-native-vs-react-native-examining-performance-31338f081980.

[6] Diestel, R. *Graph Theory*. 5. vyd. Heidelberg: Springer-Verlag, 2016. Graduate Texts in Mathematics. Sekce 1.1: Graphs. Dostupné z: https://www.math.uni-hamburg.de/home/diestel/books/graph.theory/preview/Ch1.pdf. ISBN 978-3-662-53621-6.

[7] Diestel, R. *Graph Theory*. 5. vyd. Heidelberg: Springer-Verlag, 2016. Graduate Texts in Mathematics. Kapitola 5: Colouring. Dostupné z: https://www.math.uni-hamburg.de/home/diestel/books/graph.theory/preview/Ch5.pdf. ISBN 978-3-662-53621-6.

[8] Flutter team, G. *Flutter - Frequently asked questions* [online]. 5. 12. 2018. 8. 5. 2020 [cit. 17. 5. 2020]. Dostupné z: https://flutter.dev/docs/resources/faq.

[9] Johnson, D. S. The NP-completeness column: an ongoing guide. *Journal of Algorithms*. 6. vyd. 1985, č. 3, s. 434 – 451. Dostupné z: https://doi.org/10.1016/0196-6774(85)90012-4. ISSN 0196-6774.

[10] Krogt, R. van der, Little, J., Pulliam, K., Hanhilammi, S. a Jin, Y. Scheduling for Cellular Manufacturing. In: Bessière, C., ed. *CP*. 2007. ISBN 978-3-540-74970-7.

[11] Krogt, R. van der, Little, J. a Wilson, N. *2 Context : Irish Navy Maintenance Scheduling*. 2018.

[12] Little, J. a Coughlan, B. Optimal inventory policy within hospital space constraints. *Health Care Management Science*. 1. vyd. 2008, roč. 11, č. 1, s. 177–183.

[13] Minton, S., Philips, A. B., Johnston, M. D. a Laird, P. *The min-conflicts heuristic: Experimental and theoretical results*. Scientific paper FIA-91-25. United States, 1991. Dostupné z: https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19920016200.pdf.

[14] Simonis, H., Charlier, P. a Kay, P. Constraint Handling in an Integrated Transportation Problem. *IEEE Intell. Syst.* 1. vyd. 2000, roč. 15, č. 1, s. 26–32.