VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

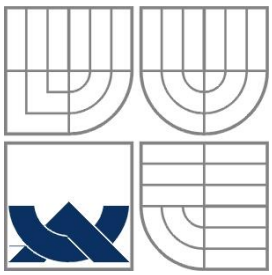
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MODUL PLÁNOVÁNÍ A ROZLOSOVÁNÍ SOUTĚŽÍ

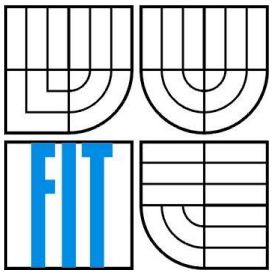
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE Zdeněk Jelínek
AUTHOR

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

MODUL PLÁNOVÁNÍ A ROZLOSOVÁNÍ SOUTĚŽÍ

PLANNING AND DRAWING MODULE FOR MATCHES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Zdeněk Jelínek
AUTHOR

VEDOUCÍ PRÁCE — Ing. Zbyněk Křivka, PhD.
SUPERVISOR

BRNO 2015

Abstrakt

Tato bakalářská práce popisuje návrh a implementaci modulu k rozlosování turnajů v kompetitivních sportech. Modul páruje hráče tak, aby bylo minimalizováno opakování zápasů, páry soupeřů hrály co nejkvalitnější zápasy, a zároveň, aby si každý hráč zahrál příslušný zápasů, a to s polynomiální nejhorší asymptotickou časovou složitostí.

Abstract

This bachelors thesis consists of description of a design and implementation of a module for drawing tournaments in competitive sports. The goal of the module is to match competitors in such a way that the repetitions of matches are minimal and the quality of the individual matches is the best, and that each competitor plays appropriate number of matches. The worst-case asymptotic time complexity of the resulting drawing algorithm is polynomial.

Klíčová slova

Vícekriteriální optimalizace, rozlosování soutěží, algoritmy a datové struktury

Keywords

Multi-objective optimization, tournament drawing, algorithms and data structures

Citace

Jelínek Zdeněk: Modul plánování a rozlosování soutěží, bakalářská práce, Brno, FIT VUT v Brně, 2015

Modul plánování a rozlosování soutěží

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Zbyňka Křivky, Ph.D.
Další informace mi poskytl Ing. Tomáš Juříček.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Zdeněk Jelínek
20. 5. 2015

Poděkování

Děkuji svému vedoucímu Ing. Zbyňku Křivkovi, Ph.D., za četné konzultace a pomoc v nasměrování mého úsilí. Jeho nadhled pro mě byl velkým přínosem obzvlášť v organizaci práce a prioritizování jednotlivých úkolů. Dále děkuji Ing. Tomáši Juříčkovi, externímu zadavateli z Multiligy, za možnost hlouběji pochopit systém Multiligy a jeho cílů v souvislosti s modulem sloužícím k rozlosování zápasů. Děkuji mu také za dodání testovacích dat, bezproblémovou spolupráci při nasazování výsledného programu a také za možnost ozkoušet všechny varianty párovacích algoritmů nad reálným vzorkem hráčů. Spolupráce s ním pro mě byla podnětná a přínosná i z hlediska praktických programátorských znalostí a schopností.

Obsah

1 Úvod

Tato bakalářská práce se zabývá popisem návrhu a implementace modulu pro rozlosování zápasů pro soutěž Multiliga[1]. Jedná se o amatérskou ligu v kompetitivních sportech, jako např. badminton, tenis, squash.

Samotné rozlosování je vícekriteriální optimalizační problém. Tato bakalářská práce popisuje, jak je pomocí specializace možné tento NP-těžký problém řešit v polynomiálním čase s $O(n)$ paměťovou složitostí při subjektivně přijatelné míře odklonění od optima.

Abychom mohli zavést patřičnou specializaci algoritmu, který slouží k rozlosování zápasů, a odvodit další mechanismy optimalizace, je vhodné se nejprve seznámit s kontextem, ve kterém bude toto rozlosování probíhat. Kapitola 2 této bakalářské práce tedy slouží k seznámení se s fungováním systému Multiligy a s požadavky, které jsou na výsledné rozlosování zápasů kladeny.

V dalším textu tato bakalářská práce metodou shora dolu popisuje jednotlivé mechanismy, které slouží k naplnění těchto požadavků. Je potřeba si uvědomovat, že některé z požadavků mohou jít ve svém naplňování proti sobě, tedy že uspokojováním jednoho požadavku způsobíme nárůst v neuspokojenosti u jiného požadavku. V takovém případě je potřeba volit, který z požadavků má mít přednost, případně v jaké míře se mají tyto požadavky vyrovnávat v naplňování.

Jde o situaci typickou pro problémy vícekriteriální optimalizace. Důsledkem této okolnosti je fakt, že v mnoha případech nelze vytvořit řešení optimální ze všech hledisek. Přibližováním se optimu z jednoho hlediska se zase optimu z jiného hlediska vzdalujeme. Z tohoto důvodu nebudeme klást silný důraz na optimalitu, i když se ji rozhodně nebudeme bezdůvodně vzdávat.

Po popisu požadavků a základních způsobů jejich naplnění následuje návrh, kde je tento popis transformován do podoby bližší programu. Návrh nastiňuje základní rysy vzniklého algoritmu a zabývá se také vhodnou reprezentací dat, zejména z hlediska konkrétního řešeného problému.

Návrh není příliš popisný z hlediska výsledného programu jako celku. V tomto ohledu pouze specifikuje rozložení jednotlivých podmodulů do hierarchie, díky které bude možné program použít ať už jako samostatnou aplikaci, nebo jako knihovnu pro použití v rámci jiné zastřešující aplikace.

V kapitole o implementaci, která následuje po návrhu, je blíže popsán výsledný program. Jedná se však spíše o popis význačných míst programu, než o detailní rozbor všech jeho dílčích částí. Důraz je zde kladen na výsledky experimentů a aplikaci programu v praxi. Jsou zde také prezentovány principy, které byly využity za účelem dosažení maximální konfigurovatelnosti a rozšiřitelnosti programu.

Protože u tohoto problému dochází s rostoucí obecností řešení k velmi rychlému nárůstu spotřeby času i paměti, tato bakalářská práce se zabývá obecností jen v malé míře. Všechny odklonění od obecného přístupu jsou v textu zdokumentovány, na některých místech jsou nastíněny i alternativy, které by bylo možné zvolit pro jiné druhy problémů.

[1] http://multiliga.cz

1.1 Definice pojmů

Abychom předešli nedorozuměním a vyhnuli se zdlouhavým opakujícím se slovním spojením, zavedeme si některé základní pojmy, které nemusí být na první pohled evidentní.

1.1.1 Složitost

Výpočetní složitost vyjadřuje náročnost řešení daného problému na z hlediska potřebných prostředků v závislosti na délce vstupních dat [1]. Běžnými prostředky jsou procesorový čas a operační paměť. U procesorového času pak hovoříme o časové složitosti, u operační paměti pak hovoříme o paměťové či prostorové složitosti.

Pro posouzení složitosti algoritmu je tedy podstatná délka vstupních dat n. Ta se liší podle řešeného problému. V obecných případech je vhodné uvažovat n blížící se k nekonečnu. Tato asymptotická složitost popisuje chování algoritmu pro vstupní data o velké délce.

Asymptotickou složitost s daného algoritmu tak můžeme vyjádřit jako funkci délky vstupních dat $s = f(n)$. V závislosti na tom, jaké má tato funkce vlastnosti, jsou definovány různé notace.

Notace velké omikron $O(f(n))$ označuje množinu všech funkcí $g_1, g_2, \ldots$ takových, pro které platí, že pro danou konstantu c je jejich hodnota v bodě $n \geq n_0$ vynásobená touto konstantou menší, než $f(n)$:

$\exists c > 0, n: n > n_0, \forall g \in O(f(n)) \mid g(n) \leq cf(n)$ [2]

Neformálně je tedy $f(n)$ od určitého bodu n_0 jakýmsi zastřešením funkce $g(n)$ z hlediska funkčních hodnot, tedy asymptotické složitosti. Mimo to jsou definovány také notace $\Omega(f(n))$ a $\Theta(f(n))$. Neformálně představuje $\Omega(f(n))$ obdobně jako $O(f(n))$ hranici. Jde ovšem o minimální hranici funkčních hodnot. A konečně neformálně $\Theta(f(n))$ představuje ohraničení danou funkcí $f(n)$ jak shora, tak také zdola, a to s potenciálně různým koeficientem pro horní a dolní hranici [2].

Navíc můžeme mluvit také o nejlepší, průměrné a nejhorší složitosti. Tyto pojmy jsou odvozené od předpokládaného rozložení vstupních dat, a tedy pravděpodobnosti, že budou tato data určitým způsobem uspořádána. Asymptotická nejlepší složitost připadá na ideální uspořádání vstupních dat. Například pro hašovací tabulku jde o situaci, kdy se hašovací funkce pro daná vstupní data přiřadí každému záznamu unikátní klíč. V takovém případě je pak přístup k datům z hlediska času konstantní. Asymptotická nejlepší časová složitost přístupu k položce v hašovací tabulce je tedy O(1).

Asymptotická nejhorší složitost pak představuje opačnou situaci. U hašovací tabulky by to znamenalo zřetězení všech n položek pod stejným klíčem. V takovém případě by pak bylo nutné pro nalezení dané položky projít až n zřetězených položek. Asymptotická nejhorší časová složitost přístupu k položce v hašovací tabulce je tedy $O(n)$.

Asymptotická průměrná složitost je pak založena na průměrném vzorku vstupních dat. Pro její určení je potřeba mít konkrétní představu o rozložení vstupních dat, a i v takovém případě nemusíme dosáhnout jejího přesného určení [3].

Dalším přístupem k výpočtu složitosti je amortizace. Asymptotická amortizovaná složitost je pak odvozena z asymptotické nejhorší složitosti pro opakující se operaci [3]. Zde je vhodným příkladem dynamické pole, které má velikost n. Při jeho naplnění n prvky je potřeba vytvořit nové vnitřní pole o délce $2n$, všechny stávající prvky překopírovat do nového vnitřního pole a původní vnitřní pole zrušit. To znamená, že prvních $n - 1$ vložení do pole má konstantní asymptotickou nejhorší časovou složitost, protože proběhne vždy jen jedna operace, kterou je zápis do pole. Každé n-té vložení pak způsobí $n - 1$ operací kopírování původních prvků a vložení n-tého prvku. Během n vložení tedy vykonáme n operací zápisu do pole a $n - 1$ operací kopírování. To znamená $2n - 1$ operací na n vložení, což odpovídá $\frac{2n-1}{n}$ operacím na jedno vložení. Pro dostatečně velké n (to plyne z faktu, že se jedná o asymptotickou

složitost) pak platí, že asymptotická amortizovaná časová složitost jednoho vložení prvku do dynamického pole je $\lim_{n\to\infty} \frac{2n-1}{n} = 2$, tedy konstantní.

1.1.2 Pár

Pro kompetitivní sporty je typické, že v daném zápase proti sobě hrají dvě entity. Těmito entitami mohou být buď jednotliví hráči, nebo týmy. V této práci budeme entitu účastnící se zápasu nazývat *hráč* za účelem lepší pochopitelnosti a intuitivnosti. Jedná se pouze o záměnu termínů bez ztráty obecnosti.

Každý zápas se tedy odehrává mezi párem soupeřících hráčů. Pár je představován množinou $\{a, b\}$, kde a a b jsou účastnící se hráči, a platí $a \neq b$.

Dalším použitým termínem je *soupeř*. Jde o hráče, který je v páru s daným hráčem. O páru $\{a, b\}$ tedy můžeme říct, že hráč a je soupeřem hráče b nebo že hráč b je soupeřem hráče a. Jde tedy o symetrický vztah. Pokud vytvoříme pár z hráče a a jeho soupeře b, neznamená to, že v tomto páru je jeden hráč a jeden soupeř. Naopak jde o dva hráče, kteří jsou vzájemnými soupeři. Lze pak také hovořit o *množině soupeřů daného hráče*, což je množina všech hráčů, kteří jsou pro daného hráče soupeři.

Pokud bude text pojednávat o *prohození soupeřů mezi páry*, jedná se o operaci, kdy z párů $\{a, b\}$ a $\{c, d\}$ vytvoříme páry $\{a, c\}$ a $\{b, d\}$, nebo $\{a, d\}$ a $\{b, c\}$. Významem je tedy to, že soupeřské vztahy v původních párech zanikají, a vznikají nové soupeřské vztahy, které jsou od těch původních různé. Hráč z prvního páru si tak vlastně prohodí soupeře s hráčem z druhého páru.

2 Analýza problému

Tato kapitola popisuje základní informace o fungování systému Multiligy. Jak a proč je vhodné párovat hráče a co potenciálně znamenají atributy užité pro párování pro výslednou hru.

Následně jsou rozvedeny některé základní způsoby, jak lze toto párování algoritmicky provádět, a jejich výhody a nevýhody. Důraz je kladen především na kvalitu výsledku, škálovatelnost, možnosti rozšiřování, konfigurovatelnost a udržovatelnost algoritmů.

2.1 Systém Multiligy, požadavky

Na Multilize se registrují uživatelé, kteří se pak přihlašují pro jednotlivé měsíce a sporty do turnajů. Každý měsíc je odehráno jedno kolo v každém sportu. Na toto kolo je třeba rozlosovat zápasy pro přihlášené hráče.

Každý hráč si před daným kolem zvolí počet zápasů, který chce odehrát. Po tom, co je kolo rozlosováno, se hráči musejí dohodnout, kde a kdy budou hrát. Po skončení hry pak do systému Multiligy zapíší a potvrdí výsledky zápasu. Dále je pro každého hráče udržována historie zápasů s ostatními hráči v předchozích kolech a Elo ohodnocení jeho schopností.

Pro algoritmus je z těchto informací stěžejní, že hráči požadují určitý počet zápasů a je potřeba, aby tento počet zápasů byl naplněn, tedy, aby si hráči zahráli tolik zápasů, kolik požadují.

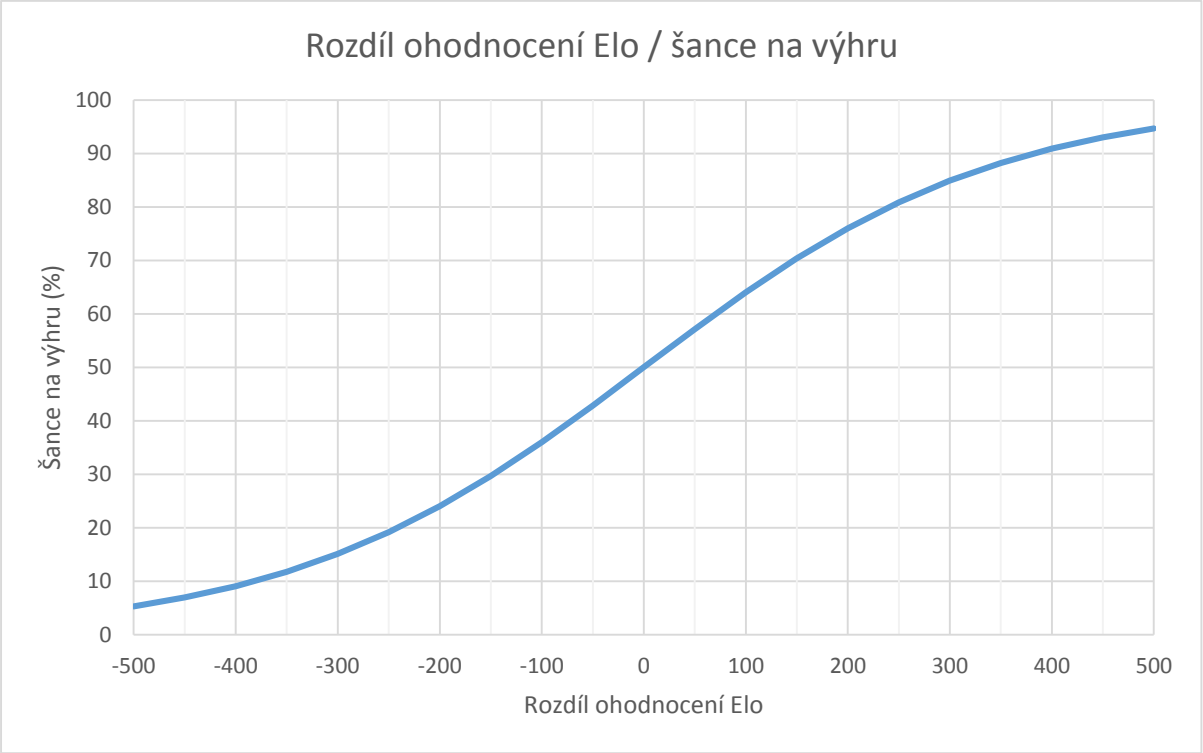
2.1.1 Elo systém

Elo systém je pojmenován po svém tvůrci, Americkém fyziku a hráči šachu, Arpadu Elovi. Tento systém slouží k teoretickému ohodnocení kvality hráče v rámci daného systému a původně se používal pro hodnocení schopností šachistů.

Každému hráči v systému je přiděleno číselné ohodnocení, které se při výhře zvyšuje a při prohře snižuje. Velikost této změny je odpovídá rozdílu ohodnocení soupeřů v příslušném zápase. Rozdělení ohodnocení mezi hráči odpovídá normálnímu rozdělení $N(1500, 200^2)$. Další informace o systému Elo, jeho implementacích a rozšířeních uvádí Glickman [4].

Z rozdílu ohodnocení Elo lze dopočítat předpokládané šance na výhru obou hráčů. Pro hráče se shodným ohodnocením jsou šance na výhru 50:50. Při rozdílu ohodnocení 100 bodů jsou předpokládané šance na výhru 65:35. Při rozdílu ohodnocení 200 bodů už jsou šance na výhru 76:24 (viz obr. 1.1). Tento rozdíl ohodnocení lze tedy využít pro hledání maximálně vyrovnaných soupeřů a platí, že čím menší rozdíl, tím vyrovnanější by soupeři měli být.

Vyrovnanost soupeřů ale nemusí být hráči vnímána pozitivně. Mnoho hráčů může naopak preferovat občasné utkání s někým na mnohem vyšší úrovni, a jindy zase s někým mnohem slabším. Někdy si povaha sportu či hry přímo diktuje nevyrovnané soupeře, jindy je nevyrovnanost zápasů součástí politiky společnosti v pozadí. Mnoho her oslovuje hráče právě kvůli své obtížnosti, náročnosti, zatímco jiné hry oslovují hráče tím, že jim umožňují trávit čas mimo jejich běžnou realitu bez nutnosti hru prožívat, zlepšovat se v ní.



Obr. 1.1 Šance na výhru vztažená k rozdílu Elo ohodnocení hráčů

2.1.2 Požadavky

Hráči by měli být na každé kolo spárováni tak, aby mezi soupeři byl co nejmenší rozdíl ohodnocení Elo a zároveň by měl být naplněn požadovaný počet zápasů u co nejvyššího počtu hráčů. Hráči se společnou historií zápasů by spolu v nejbližších kolech neměli hrát. V případě, že to je nezbytné, jsou nejprve voleni soupeři, s kterými daný hráč hrál před nejdelší dobou. Dva hráči spolu v jednom kole mohou odehrát nanejvýše jeden zápas.

2.1.3 Původní algoritmus

Před aplikací výsledků této práce byl se na Multilize používán algoritmus pro rozlosování, který byl nedostačující zejména z hlediska kvality.

Při aplikaci původního algoritmu jsou administrátoři systému nuceni ručně upravovat výsledky, protože některé vzniklé páry jsou velmi nekvalitní. Se zvyšujícím se počtem hráčů účastnících se soutěže se rovněž zvyšuje časová náročnost těchto úprav. Navíc v době psaní této práce probíhá správa systému Multiligy ve volném čase administrátorů, tedy po tom, co přijdou z práce, nebo o víkendech.

Navíc se administrátoři Multiligy obávají, že původní algoritmus nebude výkonově zvládat větší počty hráčů, se kterými do budoucna ve svém systému počítají.

Základními cíli této práce je tedy jak co nejvyšší kvalita vzniklého rozlosování, tak také vysoká výkonnost a škálovatelnost vzniklého algoritmu.

2.2 Analýza požadavků

V předchozí podkapitole bylo nastíněno několik požadavků. Nyní provedeme jejích podrobnější rozbor a klasifikaci. Objasníme, jak do sebe tyto požadavky zapadají, kde se překrývají, a za jakých okolností jsou některé z těchto požadavků protichůdné.

Rozlosování zápasů pro dané kolo v daném sportu, tedy výsledek algoritmu, budeme reprezentovat množinou párů, které představují jednotlivé zápasy. Tato informace stačí k jednoznačného popisu toho, kdo s kým má v daném kole hrát. Pořadí zápasů a další okolnosti nejsou pro fungování algoritmu podstatné.

Nejstriktnější a zároveň nejjednoznačnější požadavek je pravidlo, že se v daném kole nesmí opakovat zápas stejného páru hráčů. Tohoto cíle jde velmi snadno dosáhnout tak, že se při rozlosování stejný pár podruhé nevytváří. V jakékoli fázi provádění algoritmu můžeme jednoznačně rozhodnout, zda je dosažené řešení z tohoto hlediska validní či nikoliv. Jeho naplnění dále plyne z předpokladu, že výsledkem algoritmu je množina.

Požadavky na maximální počet párů a na naplnění co největšího počtu z požadovaných zápasů hráčů budou naplňovány průběžným vykonáváním algoritmu a opravdovou míru jejich splnění zjistíme až ve chvíli, kdy je celé rozlosování skončeno. V takové situaci můžeme dvě rozlosování porovnat mezi sebou a říci, které z řešení je lepší z hlediska počtu spárovaných hráčů nebo naplnění počtu požadovaných zápasů.

Požadavek na kvalitu zápasů je naplňován jak pro celek hráčů, tak i pro jednotlivé páry. Zde vzniká první konflikt požadavků. Pokud pro daného hráče *a* nalezneme optimálního protivníka *b*, může to v důsledku jejich požadavků na počty zápasů znamenat, že některý jiný hráč *c*, pro kterého by bylo spárování s jedním z těchto hráčů *a, b* optimální, už nedosáhne optimálního spárování, protože s nimi už nebude moct být spárován. Zde nastupuje potřeba rovnováhy mezi uspokojením jednotlivců a uspokojením celku. Je tedy potřeba, aby byli spokojeni všichni hráči jako celek, a zároveň, aby byl každý hráč co nejspokojenější jako jednotlivec.

Nelze ale říct, že jedno z těchto kritérií je jednoznačně důležitější než to druhé. Představme si například hráče *k*, který podává výjimečně kvalitní výsledky. Pro takovéhoto hráče bude jen obtížné sehnat kvalitního protivníka. V případě, že tomuto hráči přesto najdeme ty nejvhodnější protivníky *i, j*, může to znamenat, že jsme zhoršili celkovou kvalitu rozlosování, protože tito hráči *i, j* byli nejvhodnějšími protivníky také pro jiné hráče *m, n*, kteří v důsledku toho budou nuceni hrát s méně vhodnými protivníky *o, p*. A tito hráči *o, p* byli nejspíše dalšími nejlepšími protivníky pro někoho dalšího, a tak se celý tento pokles kvality zápasů nabaluje a může v konečném důsledku ovlivnit celý systém.

V takovémto případě vstupuje do konfliktu navíc kritérium pro maximální počet spárovaných hráčů a naplnění požadovaného počtu zápasů daného hráče. Stojíme tedy před rozhodnutím, zda danému hráči *k* umožnit zápas. Tím zvýšíme počet párů a lépe naplníme počet požadovaných zápasů tohoto hráče. Naopak tím ale tomuto hráči, jeho soupeřům a potenciálně i všem dalším hráčům s nejvyšší pravděpodobností přineseme horší zážitky ze hry.

Další konfliktní kritérium je požadavek na neopakování zápasů z předchozích kol. Pokud existuje pár, který je z hlediska kvality zápasu i z hlediska počtu vzniklých párů a naplněných zápasů hráče optimální, může se stát, že tento pár spolu hrál v některém z minulých kol. V určitých situacích se může stát, že si spolu nezahrají dva hráči, kteří spolu třeba hráli před delší dobou, a opakování zápasu by jim v ničem nevadilo. Nebo naopak může docházet k tomu, že se budou soupeři z těchto důvodů velmi často opakovat, protože jediná změna od nedávného kola, která je nejčastěji v podobě Elo ohodnocení, nemusí být dostatečná k nalezení vhodnějších soupeřů pro tyto hráče. To platí zejména pro hráče na okrajích rozložení, kde jsou počty hráčů výrazně menší.

Kritéria, která se navzájem v určité míře vylučují, jsou typická pro vícekriteriální optimalizaci. Určitou operací můžeme výsledek z daného hlediska zlepšit, zatímco z jiných hledisek se tím výsledek zhorší. Je proto potřeba stanovit buď priority jednotlivých kritérií, nebo váhy, podle kterých se můžeme rozhodnout, zda se zhoršení z hlediska jednoho kritéria výměnou za zlepšení z hlediska jiného kritéria vyplatí. To bude rozebráno v následující podkapitole.

Formálně se tedy pokoušíme maximalizovat kardinalitu množiny M vzniklých párů, která reprezentuje výsledné rozlosování, kde pro každého hráče p platí, že počet jeho požadovaných zápasů r_p je větší nebo roven jeho počtu naplněných zápasů m_p. Jelikož jsou další kritéria formálně obtížně zapsatelná, budeme v tuto chvíli uvažovat jejich zobecnění na vektor funkcí $[f_0, f_1, \ldots, f_n]$, kde každá z těchto funkcí představuje dílčí ohodnocení některého z kritérií nad množinou hráčů a výsledným rozlosováním. Jejich bližší popis bude v následující podkapitole.

Nezáleží na tom, zda budeme tento vektor minimalizovat, nebo maximalizovat, protože hodnotící funkce lze snadno použít k hledání opačného extrému pomocí jejich převrácené hodnoty s případným přičtením konstanty. Zvolíme například minimalizaci.

Samotná formulace problému je tedy:

Maximalizuj $|M|$ podmíněnou $r_p \geq m_p$ a
minimalizuj $[f_0, f_1, \ldots, f_n]$.

2.3 Vyhodnocení problému

Dopěli jsme k tomu, že sice známe kritéria pro optimalizaci, ale k jednoznačnému řešení problému je potřeba mezi těmito kritérii zavést určité vztahy. K zavedení těchto vztahů je však potřeba mít vytvořenou konkrétní představu o fungování jednotlivých funkcí, které slouží k ohodnocení příslušných kritérií.

Tato podkapitola detailněji popisuje hodnotící funkce a vztahy mezi nimi. Diskutuje vhodnost různých metod v kombinaci s hodnotícími funkcemi k provádění algoritmu a nastiňuje základní rysy samotného párovacího algoritmu.

2.3.1 Hodnotící funkce

Z požadavků vyplývá, že se pokoušíme naplňovat konfliktní kritéria pro celkovou kvalitu zápasů, kvalitu jednotlivých zápasů, co nejvyšší počet spárovaných hráčů, co nejvyšší počet naplněných zápasů a neopakování zápasů proběhlých v minulých kolech.

Kromě rozdělení na kritéria týkající se celku a kritéria týkající se jednotlivců se nabízí také rozdělení na kritéria, která se týkají kvality zápasů, a kritéria, která se týkají naplňování počtu zápasů. Pro tyto dva druhy kritérií zavedeme dvě hlavní hodnotící funkce.

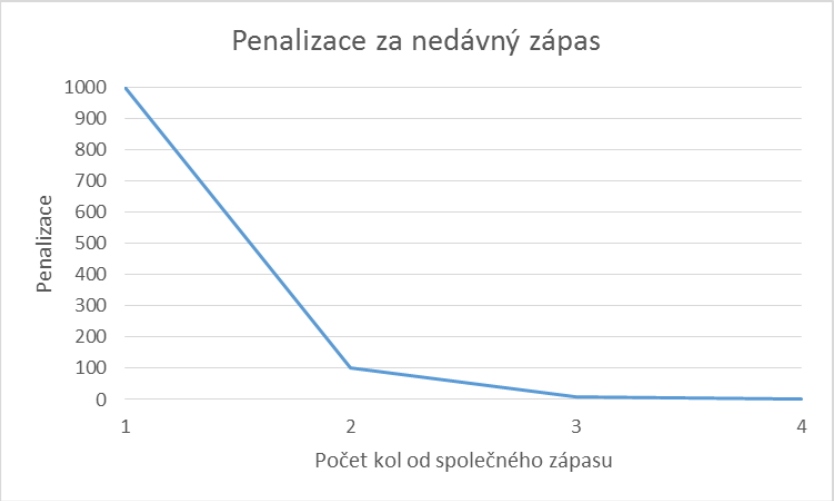
První hodnotící funkci nazveme *celkové ohodnocení kvality zápasů*, které pro výsledné rozlosování spočte předpokládanou kvalitu zápasů. Toto ohodnocení v sobě bude zahrnovat celkovou kvalitu zápasů, kvalitu jednotlivých zápasů a také míru opakování nedávných zápasů.

Druhou hodnotící funkci nazveme *celkové ohodnocení spárovaných hráčů*. Argumenty této funkce jsou výsledné rozlosování a množina všech hráčů. Tato hodnotící funkce v sobě zahrnuje kritéria pro co nejvyšší počet spárovaných hráčů a co nejvyšší počet naplněných zápasů.

2.3.1.1 Celkové ohodnocení kvality zápasů

Vyrovnanost daného jednotlivého zápasu lze jednoduše zjistit jako rozdíl Elo hodnocení soupeřů. Elo ohodnocení je pro každého hráče uloženo přímo v systému Multiligy.

K ohodnocení daného zápasu z hlediska opakování v nedávné minulosti je potřeba znát historii zápasů obou hráčů. Ta je rovněž dostupná v systému Multiligy. Zjistíme z ní, před kolika koly spolu hráči naposledy hráli, a tuto hodnotu vhodně číselně ohodnotíme jako penalizaci za nedávný zápas. Je potřeba se držet požadavku, že opakování stejného zápasu ve dvou kolech po sobě je velmi nevhodné, ale následně míra penalizace velmi rychle klesá s rostoucím počtem kol mezi opakováními (obr. 2.1).

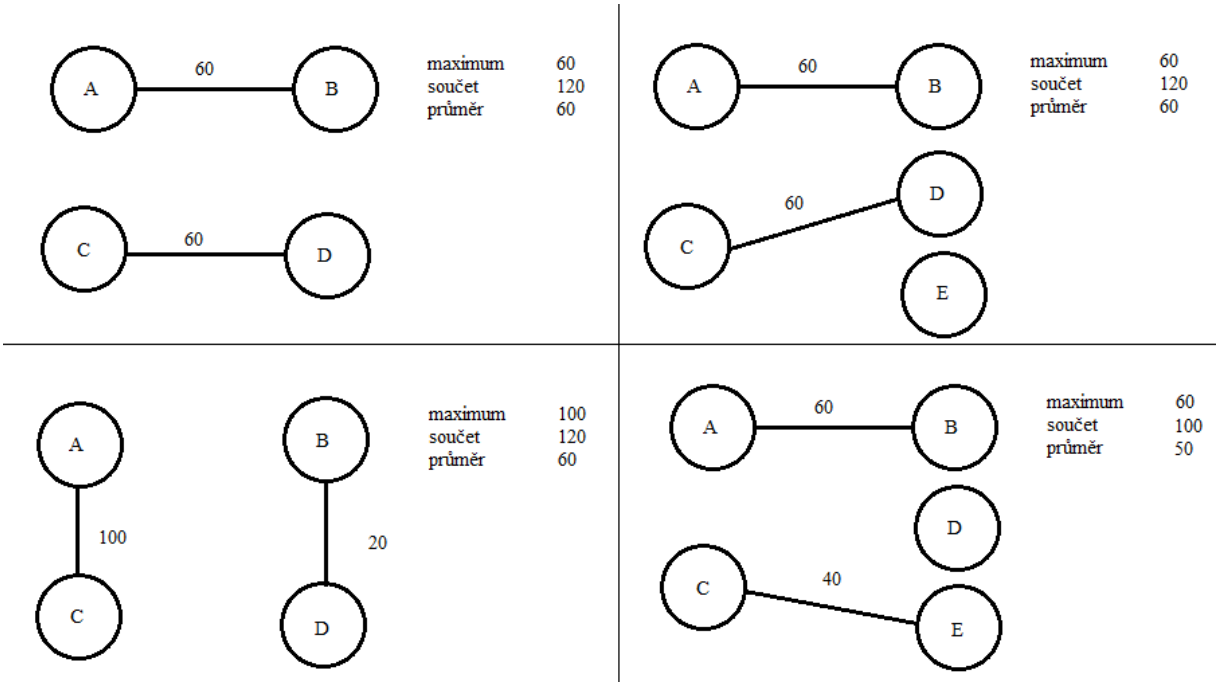


Obr. 2.1 Penalizace za nedávný zápas

Jak penalizace za nedávný zápas, tak také rozdíl Elo ohodnocení soupeřů, jsou optimální v minimu. Jejich součtem dostaneme *ohodnocení kvality jednotlivého zápasu*. Mezi rozdílem Elo ohodnocení soupeřů a penalizací za nedávný zápas je implicitní poměr vah, který je dán základem, koeficienty a konstantami exponenciální funkce pro výpočet penalizace za nedávný zápas.

Do tohoto ohodnocení lze přidávat s váhami další rozměry jako například preferenci pohlaví soupeře nebo rozdíl spolehlivosti hráčů.

Celkové ohodnocení kvality zápasů pak spočteme jako maximum z kvalit jednotlivých zápasů. Platí, že čím větší ohodnocení, tím nižší kvalita, takže maximum vyjadřuje ohodnocení nejméně kvalitního zápasu. Optimalizací maxima navíc zajistíme rovnoměrnou kvalitu zápasů díky jeho velmi nízké statistické odolnosti. Ostatní agregační funkce vykazují z opačného důvodu velké výkyvy v rovnoměrnosti hodnot (viz obr. 2.2). Z obdobných důvodů nejsou vhodné ani funkce modus a medián. Některou z dalších agregačních funkcí lze využít jako vedlejší kritérium s nižší váhou pro přesnější porovnání dvou rozlosování, která mají shodná maxima.



Obr. 2.2 Srovnání agregačních funkcí

2.3.1.2 Celkové ohodnocení spárovaných hráčů

Kritéria pro co nejvyšší celkový počet zápasů a co nejvyšší počet naplněných zápasů pro každého hráče nejsou konfliktní. Naopak sledují stejný cíl. Pokud spárujeme dva hráče, jednak jsme zvýšili celkový počet zápasů, a navíc jsme zvýšili i jejich počty naplněných zápasů. Kritérium pro celkový počet zápasů můžeme tedy úplně vynechat, protože jeho optimalizaci zajistí optimalizace kritéria pro co nejvyšší počet naplněných zápasů pro každého hráče.

Ohodnocení počtu naplněných zápasů hráče založíme na dvou základních vlastnostech. V prvé řadě je důležité, aby byl hráči naplněn počet požadovaných zápasů. Je však rozdíl mezi tím, když hráč požadující čtyři zápasy má jeden nenaplněný požadovaný zápas, a když se v takovéto situaci ocitne hráč požadující jeden zápas, takže si v daném kole vůbec nezahraje. Za tímto účelem budeme používat poměr nenaplněných zápasů k počtu požadovaných zápasů s tím, že čím je tento poměr menší, tím je ohodnocení lepší.

Další podstatnou vlastností je doba od posledního rozlosování, při kterém danému hráči nebyl naplněn požadovaný počet zápasů. Ta by měla být optimální ohodnocení co největší. V systému Multiligy však není možné tuto dobu jednoznačně určit, a tak se jí nebudeme dále zabývat. Stačilo by ji však s váhou přičíst k poměru nenaplněných zápasů k počtu požadovaných zápasů daného hráče.

K *ohodnocení počtu naplněných zápasů* h_a *hráče* a tedy potřebujeme znát jeho počet požadovaných zápasů r_a a počet jeho naplněných zápasů m_a. Připomeňme, že ze základních požadavků plyne platnost $r_a \geq m_a$.

Ohodnocení h_a pak spočteme jako poměr nenaplněných zápasů u_a k počtu požadovaných zápasů r_a hráče a: $h_a = \frac{u_a}{r_a} = \frac{m_a - r_a}{r_a}$.

Takto lze říci, že hráč x požadující dva zápasy s jedním naplněným zápasem, bude mít vyšší ohodnocení než hráč y s dvěma naplněnými zápasy ze tří požadovaných:

$$h_x = \frac{u_x}{r_x} = \frac{2-1}{2} = \frac{1}{2} > h_y = \frac{u_y}{r_y} = \frac{3-2}{3} = \frac{1}{3}$$

To znamená, že nenaplnění požadovaného počtu zápasů u hráče x je za této situace horší než nenaplnění požadovaného počtu zápasů hráče y.

Zavedli jsme tedy *ohodnocení hráče z hlediska počtu naplněných zápasů*. Pro výpočet *celkového ohodnocení spárovaných hráčů* tyto hodnoty pro všechny hráče sečteme.

2.3.1.3 Nový formální zápis problému

V předchozích dvou podkapitolách jsme popsali hodnotící funkce, kterými budeme substituovat vektor hodnotících funkcí [f_0, f_1, …, f_n] z podkapitoly 2.2. Původní formální zápis můžeme nyní přepsat s využitím nových definic.

Označíme-li *celkové ohodnocení kvality zápasů* q a *celkové ohodnocení spárovaných hráčů* p, množinu všech hráčů H, požadovaný počet zápasů hráče a r_a, naplněný počet zápasů hráče a m_a, můžeme za předpokladu, že ve výsledném rozlosování se neopakují stejné páry, zapsat řešený problém jako:

Minimalizuj q a
minimalizuj p
podmíněné $r_a \geq m_a$ pro každého hráče $a \in H$.

2.3.2 Základní rysy algoritmu

Víme tedy už, co se od algoritmu očekává, a jak ohodnotit jeho výsledek. Nyní se zamysleme nad tím, jak by měl vlastně algoritmus, který naplňuje tyto požadavky, vypadat.

V počátečním stavu provádění rozlosování máme k dispozici seznam hráčů, kteří mají být spárováni, a jejich atributy. V případě vytvoření rozlosování jsme z těchto jejich atributů s to spočítat veškerá ohodnocení tohoto rozlosování. Množina reprezentující výsledné rozlosování je v počátečním stavu prázdná.

První z možností by bylo vytvořit nějaké řešení naslepo a toto řešení se pokoušet postupně zlepšovat. Zde ale narážíme na fakt, že toto zlepšování existujícího řešení je velmi náročná operace. Pokud se rozhodneme, že dva hráči *a*, *b*, kteří spolu byli spárováni, by měli být každý spárováni raději s hráči *c*, *d*, nemusíme se pohnout z místa, protože hráči *c* a *d* už mají nejspíše naplněny požadované zápasy a je potřeba některý z jejich párů rozpojit. V případě, že hráči *c* a *d* nebyli spárováni spolu, musí nutně dojít k celému řetězu rozpojení dalších párů, dokud nenarazíme na dvojici hráčů, kteří jsou spárováni spolu. Nemáme ale zaručeno, že takováto změna nám pak jako celek přinese zlepšení kvality řešení. Takovýto přístup se pak blíží k přístupu zkoušení všech přípustných kombinací, který je z hlediska spotřeby času nevhodný.

Další možností je použít *ohodnocení počtu naplněných zápasů hráče* a *ohodnocení kvality jednotlivého zápasu* jako heuristiky pro vykonávání algoritmu. Takto můžeme pracovat s informovaným iterativním algoritmem uspořádaného vyhledávání (angl. *best-first search*).

Překážkou v tomto přístupu je fakt, že tyto hodnotící funkce jsou navrženy pro ohodnocování již dokončeného rozlosování. Tím, že spárujeme vždy hráče, jehož spárování zajistí největší zlepšení z hlediska těchto ohodnocení, ještě nezaručujeme, že takovéto řešení bude nejkvalitnější možné, protože tím můžeme vytvořit negativní podmínky pro vznik dalších párů.

Mějme například čtyři hráče *a*, *b*, *c*, *d*, kteří všichni požadují po jednom zápasu a jsou ohodnoceni takto:

hráč	Elo ohodnocení	požadováno zápasů
a	1500	1
b	1490	1
c	1550	1
d	1450	1

Tabulka 2.1 Příklad ohodnocení hráčů

Nad těmito hráči lze vytvořit tři přípustná rozlosování, ze kterých dvě ilustruje tabulka 2.2. Ohodnocení je zde rovno maximálnímu rozdílu Elo ohodnocení, obdobně jako v *celkovém ohodnocení*

kvality zápasů, kde je ale navíc přičten vliv předchozích zápasů, který zde pro zjednodušení zanedbáváme.

řešení	první pár, ohodnocení	druhý pár, ohodnocení	výsledné ohodnocení
uspořádané vyhledávání	$\{\{a, b\}, 10\}$	$\{\{c, d\}, 100\}$	100
optimální rozlosování	$\{\{a, c\}, 50\}$	$\{\{b, d\}, 40\}$	50

Tabulka 2.2 Porovnání optimálního výsledku s výsledkem uspořádaného prohledávání

Výsledek získaný uspořádaným vyhledáváním je v tomto případě výrazně horší než optimální rozlosování a to právě z důvodu, že hned zpočátku vyčerpá nejvhodnější pár a pro zbývající pár nastanou velmi nevhodné podmínky.

2.3.2.1 Uspořádané prohledávání a celkové ohodnocení kvality zápasů

Jak jsme zjistili, i na velmi malém vzorku hráčů lze ukázat, že používání uspořádaného vyhledávání v kombinaci s *ohodnocením kvality jednotlivého zápasu* vede k neoptimálním výsledkům z hlediska *celkového ohodnocení kvality zápasů*. Pokud vybereme v jedné iteraci ideální pár, může nastat situace, kdy v některé další iteraci v důsledku toho budeme nuceni vybrat pár nekvalitní, protože optimální soupeři pro takovýto pár už mají naplněn požadovaný počet zápasů. Výsledným *celkovým ohodnocením kvality zápasů* je pak ohodnocení tohoto nekvalitního páru, tedy maximum ze všech dílčích ohodnocení.

V průběhu provádění algoritmu tedy nejsme schopni odhadnout, jakým způsobem se volba soupeře podepíše na konečné kvalitě výsledku. Jedinou možností optimalizace je v případě *celkového ohodnocení kvality zápasů* skutečně pouze upravování již vytvořeného řešení. Můžeme tak například vzít pár s nejvyšším ohodnocením a najít jiný pár takový, že když prohodíme soupeře mezi těmito páry, vzniknou nové páry s nižším a tedy lepším ohodnocením. Takto můžeme kvalitu sice do určité míry zlepšit, ale párů, které můžeme takto prohodit, nebude nejspíš mnoho. Pro lepší výsledek by bylo potřeba skutečně rozpojit větší množství párů a následně je pospojovat jiným způsobem nazpět. Na základě dílčích *ohodnocení kvality jednotlivých zápasů* párů, které bychom takto vytvořili, můžeme říci, jestli během tohoto rozpojování ještě stále sledujeme zlepšení oproti původnímu řešení, nebo ho naopak zhoršujeme. Pokud by některé z nových dílčích ohodnocení bylo větší než již nalezené maximum, víme, že jsme se vydali špatným směrem a že nemá smysl v rozpojování párů dále pokračovat.

V případě použití takovéto optimalizace je možné, že časová a prostorová složitost zvoleného přístupu bude příliš vysoká, protože se jím blížíme k prohledávání všech možných kombinací. To, zda takovýto přístup znemožní používání algoritmu, závisí spíše na rozsahu a povaze řešeného problému. V případě Multiligy, kde jsou v době psaní práce k rozlosování obvykle desítky hráčů, by takovýto přístup mohl být dostatečně efektivní.

2.3.2.2 Uspořádané prohledávání a celkové ohodnocení spárovaných hráčů

Na rozdíl od *celkového ohodnocení kvality zápasů*, které je maximem z dílčích ohodnocení, je *celkové ohodnocení spárovaných hráčů* součtem, a tedy i vhodným kandidátem pro metodu uspořádaného vyhledávání. Můžeme říci, že minimalizováním jednotlivých složek součtu minimalizujeme i samotný součet a tedy že minimalizace *ohodnocení počtu naplněných zápasů hráče* v průběhu provádění algoritmu je smysluplná.

Ohodnocení počtu naplněných zápasů hráče ale nemůže být samo o sobě použito jako heuristika, protože se počítá jako poměr nenaplněných požadovaných zápasů hráče k počtu požadovaných zápasů hráče a nemusí tedy jednoznačně označit přednostního hráče, přestože intuitivní upřednostnění existuje. V situaci, kdy dva hráči *a* a *b* požadují jeden a dva zápasy $r_a = 1$, $r_b = 2$ a ani jeden z nich není spárován $m_a = m_b = 0$, je jejich ohodnocení stejné:

$$h_a = \frac{r_a - m_a}{r_a} = \frac{1-0}{1} = 1,\ h_b = \frac{r_b - m_b}{r_b} = \frac{2-0}{2} = 1,\ h_a = h_b$$

Za předpokladu, že tyto hráče nespárujeme spolu, má spárování každého z nich odlišné důsledky pro další průběh provádění algoritmu. Zatímco hráče *a* spárováním vyřadíme ze seznamu hráčů zbývajících ke spárování, hráč *b* i po spárování zůstává v seznamu hráčů ke spárování v dalších iteracích a navíc pro něj nyní existuje soupeř, se kterým už nemůže být spárován. *Ohodnocení počtu naplněných zápasů hráče* tedy nevyjadřuje v plné míře hodnotu spárování hráče z hlediska vykonávání algoritmu.

Nevhodnost *ohodnocení počtu naplněných zápasů hráče* jako heuristiky ilustruje tabulka 2.3, kde navíc pro spárování preferujeme hráče s nejvyšším počtem požadovaných zápasů v situacích, kdy nelze jednoznačně rozhodnout, jakého hráče zvolit.

krok	vzniklý pár	zbývá zápasů	ohodnocení počtu naplněných zápasů hráče
poč.	{}	*a*: 1, *b*: 1, *c*: 1, *d*: 3	*a*: 1, *b*: 1, *c*: 1, *d*: 1
1	{*d*, *a*}	*a*: 0, *b*: 1, *c*: 1, *d*: 2	*a*: 0, *b*: 1, *c*: 1, *d*: 0.67
2	{*b*, *c*}	*a*: 0, *b*: 0, *c*: 0, *d*: 2	*a*: 0, *b*: 0, *c*: 0, *d*: 0.67

Tabulka 2.3 Příklad neoptimálního výsledku při použití nevhodné heuristické funkce

Je tedy potřeba, abychom se pomocí heuristické funkce vyhnuli okolnostem, které negativně ovlivňují budoucí výsledek. První z nich je situace, kdy se hráči požadující nejmenší počet zápasů párují mezi sebou. V takové situaci dochází k velmi rychlému snižování počtu potenciálních soupeřů pro ostatní hráče. Pro ostatní pak nemusí zbýt dostatek soupeřů k tomu, aby mohly být naplněny jejich počty požadovaných zápasů.

Další záležitostí negativně ovlivňující budoucí iterace je vznik neopakovatelných zápasů. Pokud některé hráče, kteří mají být ještě spárováni, už mezi sebou nelze spárovat, není snadné určit, kdo má být spárován v následující iteraci. Za tímto účelem by hráči museli být řazeni podle počtu jejich potenciálních soupeřů. Tento počet by se musel aktualizovat s každým nově vzniklým párem a v důsledku toho by značně narostla časová a nejspíše i prostorová složitost algoritmu.

Abychom se vyhnuli párování hráčů požadujících nejmenší počet zápasů mezi sebou, budeme tedy nejprve párovat hráče požadující nejvyšší počet zápasů. A abychom se vyhnuli předčasnému vzniku vazeb mezi těmito hráči, které by negativně ovlivnily následující iterace, budeme tyto hráče párovat s hráči požadujícími nejmenší počet zápasů. Tedy hráči, kteří budou za nejnižší počet iterací vyřazeni ze seznamu hráčů ke spárování.

Hráči s nejnižším počtem požadovaných zápasů by měli být spárování přednostně, takže v určité situaci bude potřeba obrátit preferenci volení hráčů ke spárování. Toho lze jednoduše docílit tím, že v případě, kdy bude počet nenaplněných požadovaných zápasů u více hráčů stejný, zvolíme hráče, který má nejméně požadovaných zápasů. Kdybychom došli před poslední iterací algoritmu do stavu popsaného tabulkou 2.4, bude ke spárování zvolen hráč *a*, protože má stejný počet nenaplněných požadovaných zápasů jako ostatní, ale má nejmenší počet požadovaných zápasů. Hráč *a* bude spárován s hráčem *b*, který má ze všech zbývajících hráčů nejmenší počet požadovaných zápasů. To odpovídá i intuitivnímu pojetí situace, kdy nejméně nevhodné je, aby si nezahrál hráč *c* jeden zápas ze tří požadovaných, oproti ostatním situacím, kdy by si hráč *b* nezahrál jeden zápas ze dvou požadovaných, nebo kdy by si dokonce hráč *a* nezahrál vůbec, přestože požadoval pouze jeden zápas, zatímco hráč *c* si zahraje přinejmenším dva.

hráč	požadované zápasy	naplněné zápasy	zápasů zbývá naplnit
a	1	0	1
b	2	1	1
c	3	2	1

Tabulka 2.4 Příklad stavu před poslední iterací algoritmu

Z toho plyne následující postup pro výběr hráčů ke spárování. Nejprve zvolíme hráče, který má nejvyšší počet nenaplněných požadovaných zápasů. V případě, že je těchto hráčů více, volíme hráče, který má nejmenší počet požadovaných zápasů. Zvoleného hráče spojíme se soupeřem, který má ze všech hráčů nejméně požadovaných zápasů a jeho počet požadovaných zápasů ještě nebyl naplněn.

krok	vzniklý pár	zbývá zápasů	nejvyšší počet nenaplněných zápasů
poč.	{}	*a*: 1, *b*: 1, *c*: 1, *d*: 3	*d*: 3
1	{*d*, *a*}	*a*: 0, *b*: 1, *c*: 1, *d*: 2	*d*: 2
2	{*d*, *b*}	*a*: 0, *b*: 0, *c*: 1, *d*: 1	*d*: 1, *c*: 1
3	{*c*, *d*}	*a*: 0, *b*: 0, *c*: 0, *d*: 0	-

Tabulka 2.5 Ilustrace nového přístupu na příkladu z tabulky 2.3

krok	vzniklý pár	zbývá zápasů	nejvyšší počet nenaplněných zápasů
poč.	{}	*a*: 1, *b*: 1, *c*: 2, *d*: 3, *e*: 4	*e*: 4
1	{*e*, *a*}	*a*: 0, *b*: 1, *c*: 2, *d*: 3, *e*: 3	*d*: 3, *e*: 3
2	{*d*, *b*}	*a*: 0, *b*: 0, *c*: 2, *d*: 2, *e*: 3	*e*: 3
3	{*e*, *c*}	*a*: 0, *b*: 0, *c*: 1, *d*: 2, *e*: 2	*d*: 2, *e*: 2
4	{*d*, *c*}	*a*: 0, *b*: 0, *c*: 0, *d*: 1, *e*: 2	*e*: 2
5	{*d*, *e*}	*a*: 0, *b*: 0, *c*: 0, *d*: 0, *e*: 1	*e*: 1

Tabulka 2.6 Ilustrace provádění párovacího algoritmu na složitějším případu bez úplného řešení

V příkladu řešeném v tabulce 2.6 docházíme k rozlosování, kde hráč *e* zůstane s jedním nenaplněným zápasem ze čtyř požadovaných. To znamená, že *celkové ohodnocení spárovaných hráčů* tohoto rozlosování je 0.25. Tento příklad nelze vyřešit tak, aby byly naplněny počty požadovaných zápasů všech hráčů. Kdyby nebyl naplněn požadovaný počet zápasů pro jiného hráče, dostali bychom vyšší ohodnocení a tedy méně kvalitní řešení. Pro nenaplnění jednoho z požadovaných zápasů by to pro hráče *d* bylo 0.33, pro hráče *c* bylo 0.5 a pro ostatní 1. Toto rozlosování je tedy z hlediska *celkového ohodnocení spárovaných hráčů* optimální.

2.3.2.3 Závěry pro algoritmus

Můžeme tedy konstatovat, že z hlediska *celkového ohodnocení spárovaných hráčů* vede použití uspořádaného prohledávání s vhodnou heuristickou funkcí k optimálním výsledkům. Z hlediska *celkového ohodnocení kvality zápasů* bude však takovéto řešení obvykle neoptimální a bude potřeba ho nějakým způsobem zlepšit, například už zmiňovanou neinformovanou technikou postupného zlepšování.

Tato technika je však obtížně aplikovatelná. Buď s ní dokážeme řešit pouze velmi jednoduché situace, kdy vyměňujeme jen soupeře v rámci dvou párů, což nemusí vést k znatelnému zlepšení výsledku. Nebo se můžeme pokusit přetvořit značnou část řešení s tím, že časové a paměťové nároky mohou být v některých situacích neúnosné.

Další z případných možností je pak připustit zhoršení kvality rozlosování z hlediska *celkového ohodnocení spárovaných hráčů* a vytvořit díky tomu kvalitnější páry, což může značně pozitivně ovlivnit *celkové ohodnocení kvality zápasů*.

Problémem tohoto přístupu je, že potřebujeme rozpojit dva páry hráčů a pak buď vytvořit jeden nový s tím, že dva hráči zůstanou bez naplnění jejich počtu požadovaných zápasů, nebo vytvořit dva nové páry. Vytvoření dvou nových párů nenarušuje *celkové ohodnocení spárovaných hráčů*, umožňuje ho však i předchozí metoda. Naopak vznik dvou hráčů s nenaplněným požadovaným počtem zápasů se jeví jako poměrně velké narušení původní optimality s ohledem na to, že dojde ke zlepšení kvality pouze jednoho zápasu. Ano, je sice možné, že tyto hráče s nenaplněným požadovaným počtem zápasů bychom mohli po rozpojení dalších párů opět spojit a vytvořit tak velmi kvalitní řešení, ale vzhledem k vzájemné provázanosti hráčů kvůli nemožnosti opakování zápasů se na to nelze spoléhat.

Při výběru soupeře pro daného hráče volíme jako soupeře hráče s nejnižším počtem požadovaných zápasů. Těchto hráčů může být větší počet a tak můžeme vybrat toho, který má s daným hráčem nejlepší *ohodnocení kvality jednotlivého zápasu*. Takto pravděpodobně nezlepšíme *celkové ohodnocení kvality zápasů*, protože na ostatní hráče pak zbydou zápasy menší kvality. Vznikne nám však jiné rozdělení zápasů, kde některé zápasy budou velmi kvalitní a jiné naopak velmi nekvalitní.

Takové rozlosování lze pak mnohem efektivněji zlepšit prohazováním soupeřů mezi páry, protože prohození malého počtu soupeřů může nad takovým rozdělením způsobit razantnější zlepšení kvality oproti situaci, kdy byly vzniklé páry zcela náhodné a v důsledku toho byly rozdíly v jejich kvalitě menší.

Tak se můžeme dostat zpět k myšlence snížení *celkového ohodnocení spárovaných hráčů* za zvýšení *celkového ohodnocení kvality zápasů*. Pokud totiž odebereme podmínku na to, aby měl zvolený soupeř nejnižší počet požadovaných zápasů během samotného provádění algoritmu, vznikne sice neoptimální řešení z hlediska *celkového ohodnocení spárovaných hráčů*, ale pomocí jiného mechanismu výběru nejvhodnějšího soupeře můžeme vytvořit rozlosování, které umožní větší míru zlepšení z hlediska *celkového ohodnocení kvality zápasů* pomocí mechanismu prohazování soupeřů mezi páry.

Na Multilize platí, že mezi počtem hráčů n a nejvyšším počtem požadovaných zápasů m je vztah $n \gg m$. U takových vzorků hráčů se ukázalo, že lze často dojít k optimálnímu řešení z hlediska *celkového ohodnocení spárovaných hráčů* i při vynechání požadavku na nejnižší počet požadovaných zápasů při volbě soupeře. V dalších úvahách tedy tento požadavek vynecháme.

3 Návrh řešení

Zjistili jsme, že použitím metody uspořádaného hledání s vhodnou heuristikou můžeme iterativně dojít k rozlosování, které bude velmi kvalitní z hlediska *celkového ohodnocení spárování hráčů*. Po jeho vytvoření se pokusíme toto rozlosování dále vylepšit z hlediska *celkového ohodnocení kvality zápasů*, což by mělo jít vzhledem k nerovnoměrnému rozložení kvality jednotlivých zápasů dobře. Naše možnosti v tomto ohledu jsou však stále omezené. Je potřeba z nich vytěžit co nejvíce bez nadměrného navyšování časové a paměťové složitosti algoritmu.

Tato kapitola popisuje detailněji fungování a vlastnosti nově vytvořeného párovacího algoritmu z hlediska datových struktur a vztahů mezi nimi. Zabývá se způsoby, jak co nejefektivněji vybírat hráče ke spárování a jejich soupeře. Popisuje také konkrétnější varianty algoritmů a jejich předpokládané vlastnosti.

V závěru tato kapitola také popisuje způsob rozdělení jednotlivých částí výsledného programu za účelem jak snadného vývoje a testování jako samostatné aplikace v prostředí osobního počítače, tak také bezproblémového nasazení do serverového prostředí jako modulu pracujícího v rámci nadřazené webové služby.

3.1 Podstata algoritmu

Samotný algoritmus párování je iterativní. V každé iteraci vybere vhodného hráče ke spárování, najde pro něj vhodného soupeře a vzniklý pár vloží do množiny reprezentující výsledné rozlosování.

Tato podkapitola se zabývá jednotlivými částmi iterace. Popisuje, jakým způsobem je vhodné vybrat hráče, který má být spárován, a jeho protivníka, s co nejmenšími výpočetními nároky. Diskutuje různé varianty a přístupy z hlediska výkonnosti algoritmu a zároveň pojmenovává limitace zvolených přístupů z hlediska kvality jimi dosaženého výsledku. V závěru je uveden pseudokód jádra algoritmu a diskutována úskalí plynoucí z jeho podoby.

3.1.1 Volba hráče ke spárování

Hráč, který má být spárován, musí splňovat dvě základní podmínky. První z nich je, že dosud nebyl naplněn jeho počet požadovaných zápasů.

To můžeme zjistit z počtu naplněných a počtu požadovaných zápasů hráče. Počet požadovaných zápasů je základní atribut a bude tedy potřeba ho uložit pro každého z n hráčů, asymptotická paměťová složitost uložení těchto počtů je tedy $O(n)$. Přestože bychom mohli počet naplněných zápasů vždy dopočítat z výsledného rozlosování, znamenalo by to v nejhorším případě $O(n)$ přístupů do kolekce reprezentující rozlosování a tedy zhoršení z hlediska času. Z tohoto důvodu je výhodnější počet naplněných zápasů také uložit jako atribut hráče. Jde sice o redundantní informace, nicméně paměťová složitost zůstává stále na $O(n)$ a časová složitost je $O(1)$.

Druhá podmínka je, že tento hráč musí mít ze všech hráčů nevyšší počet nenaplněných požadovaných zápasů. Pokud je takových hráčů více, je z nich zvolen hráč s nejnižším počtem požadovaných zápasů.

Budeme tedy operovat s množinou všech hráčů, kteří nemají naplněný požadovaný počet zápasů, a z této množiny budeme vybírat maximální prvek daný predikátem splňujícím výše uvedenou podmínku, tedy že maximální prvek reprezentuje ten hráč, který má ze všech hráčů s nejvyšším počtem nenaplněných zápasů nejnižší počet požadovaných zápasů.

K reprezentaci takovéto množiny se v algoritmech uspořádaného prohledávání obvykle používá halda. Vzhledem k tomu, že počet hráčů je předem znám, je možné použít například binární haldu v předpřipraveném poli vhodné velikosti.

Z dalších datových struktur připadá v úvahu seřazené pole a seřazený vázaný seznam. Pole oproti haldě ztrácí na straně výkonu kvůli potřebě přesouvat velké množství prvků při případné změně ohodnocení některého z prvků a následném seřazení. Prvky jsou v něm však umístěny v paměti za sebou tak, jak je algoritmus bude potřebovat, takže je možné, že se tato varianta prakticky ukáže jako časově efektivnější z důvodu lepšího využívání paměti cache. Seřazený seznam sice umožňuje rychlé vkládání, rušení a tedy i přeuspořádání, ale kvůli potřebě přistupovat náhodně do paměti během jeho procházení je pro moderní procesory za tímto účelem nevhodný.

Prozatím budeme tedy operovat s *kolekcí hráčů ke spárování*, která reprezentuje vhodně seřazenou množinu všech hráčů s nenaplněným požadovaným počtem zápasů. Tato kolekce bude představována buď haldou, nebo seřazeným polem. K rozhodnutí, kterou datovou strukturu použít, je potřeba znát další detaily ohledně fungování algoritmu. K rozhodnutí se tedy vrátíme až dále v textu.

3.1.2 Volba soupeře

Podmínky pro volbu soupeře *s* daného hráče *h* jsou rozsáhlejší, než tomu bylo u volby hráče ke spárování. V prvé řadě soupeř *s* opět nesmí mít naplněny všechny požadované zápasy. Jelikož je soupeř také hráč, všechny potřebné informace k vyhodnocení této podmínky jsou k dispozici, jak bylo uvedeno v předchozí podkapitole.

Další podmínkou pro volbu soupeře je, aby v už vygenerovaném rozlosování nebyl přítomen pár $\{h, s\}$. A navíc by tento soupeř měl být hráčem s nejlepším *ohodnocením kvality jednotlivého zápasu* vůči hráči *h*.

3.1.2.1 Vyhledání nejvhodnějšího soupeře

Vyhledání soupeře je jedním ze základních kroků algoritmu, stojí přímo v jeho jádře. Je proto nezbytné, aby tato operace byla co nejrychlejší, protože tím ovlivní celou dobu běhu algoritmu.

Řešení, kdy bychom si pro každého hráče drželi seznam potenciálních soupeřů seřazených podle vhodnosti, sice vypadá na první pohled jako velmi výhodné, ale při hlubším zamyšlení se ukazuje jako neúměrně náročné na paměť. Každý z n hráčů by měl uložen seznam potenciálních soupeřů, kterých by bylo v nejhorším případě $O(n)$. Asymptotická nejhorší paměťová složitost tohoto přístupu je $O(n^2)$, což není vhodné u algoritmu, který má být aplikovatelný na rozsáhlé problémy.

Další slabinou tohoto přístupu je, že hráčům jsou v průběhu provádění algoritmu naplňovány požadavky na počet zápasů. Hráč, který má naplněny všechny požadované zápasy, už by pro nikoho neměl být potenciální soupeř, takže při každém naplnění požadovaného počtu zápasů daného hráče je potřeba u všech jeho potenciálních soupeřů tohoto hráče odebrat z jejich seznamu potenciálních soupeřů. Pro $O(n)$ potenciálních soupeřů tedy odebíráme jednoho hráče z jejich seznamů potenciálních soupeřů, kde každý seznam má délku $O(n)$. Provedeme tedy $O(n^2)$ operací. Pokud je maximální počet požadovaných zápasů m, dojde k jeho naplnění u některého hráče v průměru jednou za $O\left(\frac{m}{2}\right)$ iterací, protože v každé iteraci jsou spárováni dva hráči. V průměru by tedy v každé iteraci bylo provedeno $O\left(\frac{2n^2}{m}\right)$ operací pro vyhledání soupeře.

Pokud místo toho uložíme všechny hráče jednoduše v pomocném poli, asymptotická nejhorší paměťová složitost bude $O(n)$ a budeme muset prohledat $n - 1$ hráčů, abychom našli vhodného soupeře. To znamená provést v nejhorším případě $O(n)$ operací v každé iteraci algoritmu pro nalezení soupeře.

Pro případy, kdy $n \geq m$, mezi které patří i Multiliga, se uložení hráčů do pole a jeho lineární prohledávání vyplatí jednoznačně lépe než používání seznamu potenciálních soupeřů uloženého u každého hráče. Platí totiž, že amortizovaná časová složitost odebrání spárovaného hráče ze seznamu u všech ostatních hráčů $O\left(\frac{n^2}{m}\right)$ je za takových okolností větší než asymptotická nejhorší časová složitost lineárního průchodu polem $O(n)$. V ostatních situacích je otázka rozsahu a povahy řešeného problému, zda je asymptotická nejhorší paměťová složitost $O(n^2)$ skutečně výhodná v porovnání se ziskem na straně času.

V dalším textu tedy budeme uvažovat práci s pomocným polem k ukládání množiny potenciálních soupeřů. Toto pole budeme označovat *pole potenciálních soupeřů*.

3.1.2.2 Nepřípustné páry

Ukládání nepřípustných soupeřů pro daného hráče je rovněž základní záležitost, nad kterou je třeba se zamyslet.

Nejjednodušším řešením by bylo použít pro uložení výsledného rozlosování kolekci, ve které lze rychle vyhledávat, abychom zjistili, zda daný pár již existuje. Uvažujme například hašovací tabulku. Pro n hráčů požadujících až m zápasů by hašovací tabulka měla $O(mn)$ asymptotickou nejhorší paměťovou složitost a $O(1)$ až $O(mn)$ asymptotickou časovou složitost jednoho vyhledání. Při hledání soupeře pro daného hráče ovšem postupně procházíme $O(n)$ potenciálních soupeřů, kde u každého z nich musíme zkontrolovat, zda už s nimi daný hráč nebyl spárován. To znamená $O(n)$ přístupů do hašovací tabulky a tedy asymptotickou časovou složitost $O(n)$ až $O(mn^2)$.

V případě, že bychom u hráčů ukládali seznam potenciálních soupeřů, dali by se nepřípustní soupeři snadno filtrovat tak, že by byli při spárování z tohoto seznamu vždy vyloučeni. Použitím takovéhoto seznamu bychom však dosáhli paměťové složitosti $O(n^2)$, která je pro řešení rozsáhlejších problémů nevhodná, jak bylo diskutováno v předchozí podkapitole.

Další alternativou je uložit výsledné rozlosování jako pole a u každého hráče držet seznam soupeřů, se kterými už byl spárován. Rozlosování jako pole má $O(mn)$ asymptotickou nejhorší paměťovou složitost. Seznam soupeřů jednoho hráče má $O(m)$ asymptotickou nejhorší paměťovou složitost. Pro n hráčů je pak asymptotická nejhorší paměťová složitost $O(mn)$, stejná jako při použití hašovací tabulky. Při hledání soupeře pak projdeme nanejvýše n možných protivníků, kde každý z nich má v seznamu nepřípustných protivníků uložených až m soupeřů. Asymptotická nejhorší časová složitost je tedy $O(mn)$.

Pokud připustíme $m \gg n$, což v případě Multiligy platí, výsledné asymptotické nejhorší paměťové složitosti jsou v obou případech $O(mn) \approx O(n)$. Asymptotická nejhorší časová složitost u hašovací tabulky je pak $O(n)$ až $O(mn^2) \approx O(n^2)$, zatímco u přístupu se seznamy nepřípustných soupeřů uložených v atributech jednotlivých hráčů je asymptotická nejhorší časová složitost $O(mn) \approx O(n)$.

Nejvýhodnějším způsobem pro zjištění, zda již daný pár existuje, je tedy v případě Multiligy ukládání seznamů soupeřů, se kterými byl daný hráč již spárován, pro každého hráče. Tento přístup má asymptotickou nejhorší paměťovou složitost $O(mn)$. Vyhledání existujících párů pro daného hráče a až n jeho soupeřů má asymptotickou nejhorší časovou složitost $O(mn)$. Zaznamenání existence nového páru má asymptotickou nejhorší časovou složitost $O(1)$.

3.1.2.3 Celková asymptotická časová složitost volby vhodného soupeře

Popsali jsme vhodné techniky pro vyhledání vhodného soupeře a zjištění, zda je pár daného hráče s daným soupeřem přípustný.

Nejvhodnějšího soupeře vyhledáme v poli. Pro *n* hráčů je jeho asymptotická nejhorší paměťová složitost O(*n*) a asymptotická časová časová složitost vyhledání O(*n*). Soupeře, se kterými již nelze daného hráče spárovat ukládáme do seznamu, který je atributem každého hráče. Pro maximální počet požadovaných zápasů *m* a počet hráčů *n* mají tyto seznamy celkovou asymptotickou nejhorší paměťovou složitost O(*mn*). Při prohledávání soupeřů pak pro každého z O(*n*) možných soupeřů prohledáváme seznam už existujících párů daného hráče, který obsahuje až *m* záznamů. Výsledná asymptotická nejhorší časová složitost je pak O(*mn*).

Celková asymptotická nejhorší časová i paměťová složitost volby vhodného soupeře je tedy O(*mn*).

3.1.3 Jádro algoritmu

Samotný algoritmus lze popsat následujícím pseudokódem:

```
1     dokud je kolekce k hráčů ke spárování neprázdná
2           odeber z kolekce k nejvhodnějšího hráče h
3
4           najdi nejvhodnějšího soupeře s pro hráče h
5           pokud soupeř s neexistuje
6                 proveď další iteraci
7
8           vytvoř a ulož pár {h, s}
9           pokud nemá hráč h naplněny všechny požadované zápasy
10                umísti hráče h zpět do kolekce k
11
12          pokud nemá soupeř s naplněny všechny požadované zápasy
13                aktualizuj ohodnocení soupeře s a seřaď kolekci k
14          jinak
15                odeber soupeře s z kolekce k
```

3.1.3.1 Důsledky pro datové struktury

Takto popsaný algoritmus má závažné důsledky pro použité datové struktury. Ty však nemusí být na první pohled zjevné. Proto nyní rozebereme operace, které algoritmus provádí.

Hned na prvním řádku se dotazujeme, zda je *kolekce obsahující hráče ke spárování* neprázdná. Tuto kolekci představuje buď seřazené pole, nebo halda. V obou případech je asymptotická nejhorší časová složitost těchto operací O(1).

Na druhém řádku odebíráme z *kolekce obsahující hráče ke spárování* maximální prvek. Tato operace má pro haldu asymptotickou nejhorší časovou složitost O(log *n*), pro seřazené pole pak asymptotickou nejhorší časovou složitost O(1) za předpokladu, že je maximální prvek na konci pole.

Na 4. řádku voláme funkci k vyhledání nejvhodnějšího soupeře. Jak bylo objasněno v předchozích podkapitolách, tato operace má asymptotickou nejhorší časovou složitost O(*mn*) při asymptotické nejhorší prostorové složitosti O(*mn*) pro *n* hráčů a maximální počet požadovaných zápasů *m*.

Tato funkce může skončit s prázdným výsledkem. To znamená, že pro daného hráče už neexistují další soupeři a tento hráč zůstane s nenaplněnými požadovanými zápasy. To reflektují řádky 5-6.

V opačném případě je vytvořen pár a umístěn do kolekce reprezentující výsledné rozlosování. Tato operace má konstantní amortizovanou časovou složitost.

Hráče ke spárování jsme na druhém řádku pseudokódu odebrali z *kolekce hráčů ke spárování*. Pokud hráč požaduje další zápasy, je třeba ho do této kolekce vložit zpět na příslušné místo. To znamená O(log n) operací u haldy a O(n) operací u seřazeného pole. Provedení této akce reprezentují řádky 9 a 10.

Na řádcích 12 a 13 vzniká komplikace pro datové struktury reprezentující *kolekci hráčů ke spárování*. Dochází zde k potřebě posunout prvek směrem dál od maxima v kolekci, protože při spárování daného hráče klesá jeho priorita pro další spárování. Pro seřazené pole to znamená tohoto hráče v poli vyhledat pomocí binárního vyhledávání s asymptotickou nejhorší časovou složitostí O(log n) a přesunout jej na odpovídající místo s asymptotickou nejhorší časovou složitostí O(n).

U hald existuje operace pro snížení hodnoty klíče (angl. *decrease-key*). Tato operace však posouvá daný prvek směrem k vrcholu haldy. My ovšem požadujeme s opačný jev, posun prvku v haldě směrem dolů. Pro haldu to znamená nutnost prvek nalézt, odebrat a znovu vložit. Nalezení prvku v haldě má asymptotickou nejhorší časovou složitost O(n), zbylé operace asymptotickou nejhorší časovou složitost O(log n).

Tato záležitost má jeden zásadní důsledek, kterým je nemožnost použít složitější a efektivnější druhy hald. Například pro Fibonacciho haldu znamená odebrání prvku vznik dvou nových podstromů a následné vložení prvku vytváří další podstrom [5]. V případě častého opakování této operace dochází k degradaci výkonu haldy, protože se skutečná časová složitost odebrání maximálního prvku začne blížit O(n) z důvodu nutnosti velmi často slévat několik podstromů vzniklých z důvodu odebírání prvků. Jediným východiskem je pro nás tedy buď použít binární haldu, nebo seřazené pole.

3.1.3.2 Důsledky pro kvalitu výsledku

Při provádění algoritmu však dochází také k jinému jevu, který ovlivňuje kvalitu výsledku. K jeho odhalení je však potřeba se zamyslet nad algoritmem z hlediska více iterací a jejich vzájemných vazeb.

Tímto jevem je chování, kdy v *kolekci hráčů ke spárování* zbydou sice hráči s nejmenší důležitostí pro naplnění jejich požadovaných zápasů, ale nemáme žádnou představu o tom, do jaké míry se k sobě tito hráči hodí do páru. Může se tak stát, že spolu skončí hráči v rozporu se základními podmínkami fungování algoritmu a celé rozlosování je pak prakticky bezcenné. Takto vytvořené extrémní páry je potřeba rozpojit a poskládat jinak. Dochází tak k dodatečnému zpracování výsledku, kde nastává výměna pozic v rámci hierarchie kritérií, kdy se snažíme optimalizovat *celkové ohodnocení kvality zápasů* při udržení stejné hodnoty *celkového ohodnocení spárovaných hráčů*.

3.1.3.3 Předpokládaná složitost algoritmu

Algoritmus během svého provádění využívá několika datových struktur umístěných na různých místech. Tyto struktury pak ovlivňují celkovou dobu jeho běhu.

Následující tabulka popisuje nejhorší asymptotické nebo amortizované časové složitosti při zvolení datové struktury binární haldy a seřazeného pole jako *kolekce hráčů ke spárování* pro počet hráčů n a maximální počet požadovaných zápasů m. Všechny uvedené složitosti byly diskutovány v předcházejících podkapitolách.

operace	seřazené pole	binární halda
kontrola neprázdnosti kolekce	O(1)	O(1)
odebrání maxima z kolekce	O(1)	O(log *n*)
výběr nejvhodnějšího soupeře	O(*mn*)	O(*mn*)
vytvoření a uložení páru	O(1)	O(1)
umístění hráče zpět do kolekce	O(*n*)	O(log *n*)
aktualizace ohodnocení soupeře	O(*n*)	O(*n*)
výsledná časová složitost jedné iterace	**O(*mn*)**	**O(*mn*)**

Tabulka 3.1 Shrnutí časových složitostí jednotlivých operací algoritmu

3.2 Varianty implementace algoritmu

V předchozí podkapitole jsme si představili jádro nového párovacího algoritmu, jeho základní vlastnosti a datové struktury, které je výhodné v něm použít.

Tato podkapitola popisuje možnosti reprezentace dat, využívání konkrétních datových struktur v algoritmu, rozličné implementace a specializace podstatných pomocných funkcí a řešení úskalí, na která jsme v minulé podkapitole narazili.

3.2.1 Algoritmus založený na grafech

Vzhledem k tomu, že všechna ohodnocení jsou předem známa, můžeme se na řešený problém dívat jako na graf, kde uzly představují hráče a hrany možná spárování mezi nimi. Takto vzniká úplný graf. Ohodnocení uzlu odpovídá *ohodnocení počtu naplněných zápasů hráče*, ohodnocení hrany odpovídá *ohodnocení kvality jednotlivého zápasu*.

Kolekce hráčů ke spárování tedy neobsahuje přímo hráče, ale uzly grafu. Ty v sobě udržují také množinu hran, které z nich vedou do dalších uzlů. V případě naplnění počtu požadovaných zápasů daného hráče jsou odstraněny všechny hrany vedoucí do uzlu reprezentujícího tohoto hráče. Daná hrana je také odstraněna při spárování dvou hráčů, kteří jsou reprezentováni uzly, které tato hrana spojuje.

To znamená, že v rámci atributů spojených s daným hráčem, je udržován seznam potenciálních soupeřů. Pokud je ke spárování zadáno n hráčů, tento seznam bude obsahovat $n - 1$ záznamů. Asymptotická nejhorší paměťová složitost jednoho záznamu je tedy $O(n)$. Vzhledem k tomu, že tento záznam je držen pro každého hráče, je celková asymptotická nejhorší paměťová složitost $O(n^2)$.

Tato zásadní nevýhoda v podobě obrovské spotřeby paměti už byla diskutována v předchozí podkapitole. Tento přístup však vykazuje vysokou úroveň obecnosti. Lze ho využít k řešení dalších podobných problémů. Pro potřeby Multiligy je z výkonostních důvodů nevhodný.

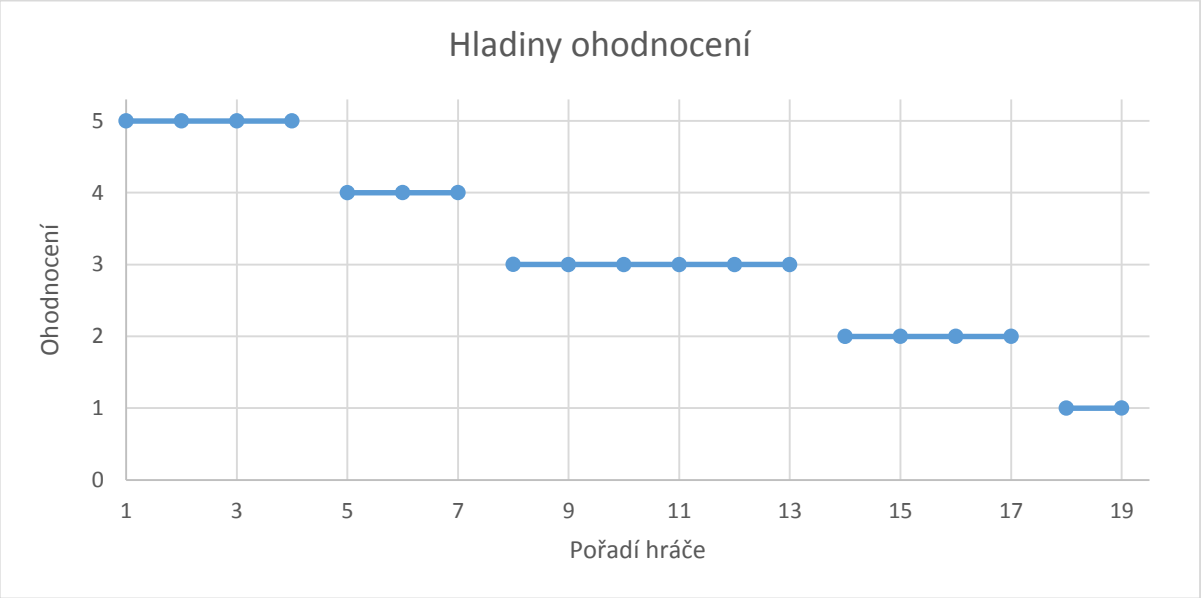
3.2.2 Specializované algoritmy

Asociace s grafem není zcela ideální způsob pohledu na řešený problém. Je to přílišné upnutí se na určitou abstrakci, která řešený problém příliš zobecňuje do podoby, kdy nelze využít konkrétních vlastností řešeného problému k optimalizaci provádění algoritmu. V případě specializovaných algoritmů se tedy oprostíme od veškerých vzorů a jednoduše budeme pracovat přímo s variantami naznačenými v kapitole 3.1 a dále rozvedeme jejich specializaci za účelem vylepšení výkonu.

3.2.2.1 Kolekce hráčů ke spárování

Kolekce hráčů ke spárování představuje množinu hráčů, jejichž počet požadovaných zápasů nebyl doposud naplněn. Z této kolekce jsou během provádění algoritmu odebírány maximální prvky, tedy hráči, kteří by měli být přednostně spárováni. Tito hráči jsou pak do kolekce navraceni, pokud ještě nemají naplněny své počty požadovaných zápasů. K vytvoření páru jsou pak v této kolekci vyhledáváni soupeři. Tito soupeři jsou po spárování buď z kolekce odebráni, nebo je potřeba upravit jejich ohodnocení a zařadit je na příslušné místo, protože spárováním jejich priorita pro další párování klesla. Tuto kolekci je možné reprezentovat binární haldou, nebo seřazeným polem.

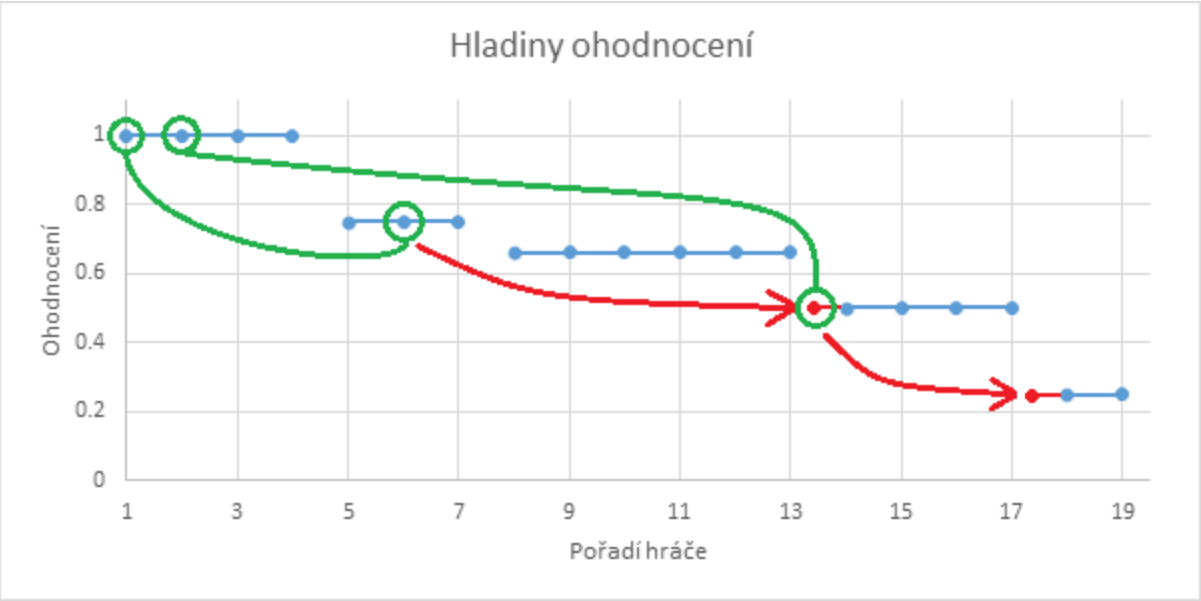
Ohodnocením hráče v této kolekci je počet nenaplněných požadovaných zápasů. Pro n hráčů, kde každý z nich požaduje až m zápasů a navíc platí $n \gg m$, budou vždy vznikat skupiny hráčů, kteří mají stejný počet nenaplněných požadovaných zápasů, a tedy budou mít i stejné ohodnocení. Takto vzniká *hladina ohodnocení*, tedy ohodnocení, které sdílí množina hráčů. *Maximální hladina ohodnocení* je pak sdílené ohodnocení hráčů na vrcholu *kolekce hráčů ke spárování*.



Obr. 3.1 Příklad hladin ohodnocení

Pokud se mezi dvěma iteracemi párovacího algoritmu nezmění *maximální hodnota ohodnocení*, znamená to, že hráč, který byl v první iteraci zvolen jako protivník, nemůže být v druhé iteraci odebrán z vrcholu kolekce jako hráč, pro kterého má být vyhledán soupeř. Je tomu tak proto, že pokud se hráč nachází na dané *hladině ohodnocení*, jeho spárováním klesne jeho ohodnocení a on se posune na nižší *hladinu ohodnocení* (viz obr. 3.2). Aby byl tento hráč odebrán z vrcholu kolekce, musela by se tedy změnit *maximální hladina ohodnocení* mezi iteracemi algoritmu, což je v rozporu s naším předpokladem.

Není tedy nutné znovu řadit hráče pokaždé, když se změní jeho ohodnocení. Naopak stačí provést řazení tehdy, když dojde ke změně *maximální hladiny ohodnocení*. Jelikož je ale možné spolu spárovat dva hráče, kteří se nacházejí na *maximální hladině ohodnocení*, může se stát, že na vrcholu kolekce se vyskytne hráč, jehož skutečné ohodnocení neodpovídá *maximální hladině ohodnocení*. Je proto potřeba před odebráním hráče kontrolovat, zda jeho skutečné ohodnocení odpovídá jeho pozici na vrcholu.



Obr. 3.2 Ilustrace průběhu párování na příkladu z obrázku 3.1. Vzniklé páry jsou označeny zeleně, posun spárovaného hráče hladinami ohodnocení směrem dolů je označen červeně. U protivníka s původním pořadím 6 došlo ke zbytečnému zařazení na hladině 0,5 po prvním spárování.

Takto ušetříme zbytečná řazení hráčů probíhající během iterací se stejnou *hladinou ohodnocení*. To se nemusí na první pohled jevit jako velký rozdíl. Efektivita tohoto přístupu roste spolu s rostoucím rozdílem mezi celkovým počtem hráčů a nejvyšším počtem požadovaných zápasů, protože se zvyšuje počet hráčů v dané hladině a tedy i počet vzniklých párů před nutností znovu provést řazení.

Zatímco růst počtu hráčů je prakticky neomezen, růst nejvyššího počtu požadovaných zápasů se na určité hodnotě zcela jistě zastaví z důvodu nemožnosti zahrát určitý počet zápasů za jeden měsíc. Obzvláště, přičteme-li k tomu fakt, že Multiliga je amatérskou ligou.

Užitím tohoto přístupu ale vzniká komplikace s využíváním binární haldy. Ta, ač se oproti seřazenému poli intuitivně jeví jako daleko zajímavější a vhodnější struktura, nám neumožňuje rozhodovat, kdy dochází k řazení prvků. Možným řešením této okolnosti by bylo vytvořit pomocné pole, kam budeme hráče po odebrání z haldy ukládat a po vyčerpání haldy použijeme toto pole jako základ pro vznik nové haldy. Uložíme si tedy hráče do pomocného pole, aby nebyli automaticky řazeni haldou, a až v případě nutnosti z nich znovu vybudujeme haldu, kterou budeme používat k dalšímu vykonávání algoritmu.

Tím se ale začíná smývat rozdíl mezi používáním haldy a seřazeného pole. Na vzniklou situaci můžeme pohlížet také tak, že máme pole hráčů, na které aplikujeme Heap sort. Tak jsme vlastně od binární haldy přešli k implementaci založené na seřazeném poli. Navíc vzhledem k tomu, že implementačním prostředím pro modul rozlosování je rámec .NET, který už sám o sobě vysoce efektivní metodu řazení obsahuje, jeví se použití haldy jako méně vhodné, protože povede jak ke komplikovanější implementaci, tak k menšímu výkonu v porovnání s řadícím algoritmem v .NETu.

Jako *kolekci hráčů ke spárování* tedy použijeme seřazené pole. Hráče nebudeme z pole průběžně odebírat, ani nijak přesouvat, jen vždy při snížení *maximální hladiny ohodnocení* odebereme z pole hráče s naplněným počtem požadovaných zápasů a pole znovu seřadíme. Jelikož nám při odebírání hráčů nezáleží na pořadí, protože bude následovat řazení, můžeme je jednoduše prohodit s dalšími hráči na konci pole s asymptotickou nejhorší časovou složitostí O(1). Samotné řazení má pak asymptotickou nejhorší časovou složitost O(n log n) stejnou, jakou by nám poskytla halda. Lze navíc předpokládat jeho velmi rychlý průběh, protože pole bude již částečně seřazené.

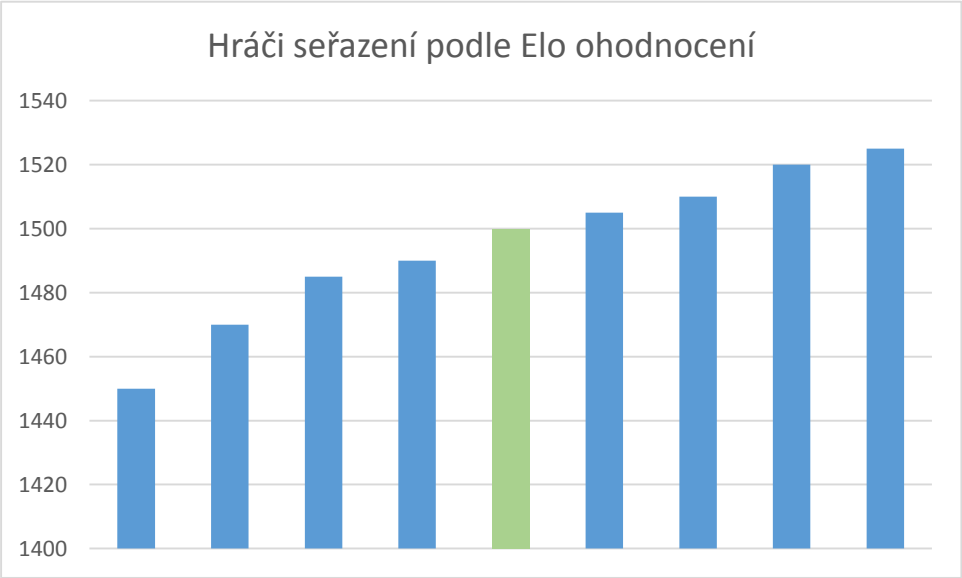
3.2.2.2 Nalezení nejvhodnějšího soupeře

Nalezení nejvhodnějšího soupeře pro daného hráče takovým způsobem, že v *poli potenciálních soupeřů* hledáme soupeře takového, který nebyl s daným hráčem spárován, má pro *n* hráčů požadujících jednotlivě O(*m*) zápasů asymptotickou nejhorší časovou složitost O(*mn*). S lineární časovou složitostí projdeme pole potenciálních soupeřů. Pro každého potenciálního soupeře pak u daného hráče zkontrolujeme, zda už se nenachází v seznamu hráčů, se kterými byl již daný hráč spárován.

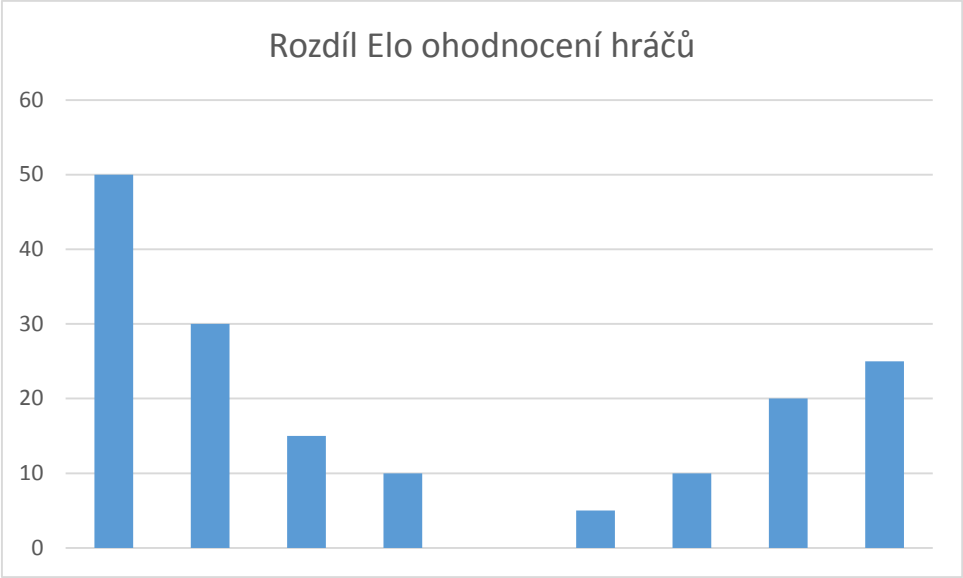
Seznam hráčů, s kterými byl již daný hráč spárován, ukládáme přímo ke hráči. Jeho velikost je limitována počtem *m* požadovaných zápasů daného hráče. Kdybychom seznam reprezentovali hašovací tabulkou, můžeme dosáhnout asymptotické časové složitosti O(1) až O(*m*). V případě Multiligy je však počet požadovaných zápasů malé číslo. Obvykle půjde o jednotky, maximálně desítky, protože lze předpokládat, že amatérský sportovec neodehraje za měsíc mnoho zápasů. V systému se doposud nejčastěji vyskytují hodnoty *m* v rozmezí 1-4. Pro takováto čísla se však nevyplatí sestrojovat hašovací tabulku, protože vzhledem k její velikosti můžeme předpokládat velké počty kolizí a tedy časovou složitost blížící se O(*m*). V případě Multiligy tedy můžeme seznam soupeřů, se kterými byl daný hráč již spárován, reprezentovat jednoduchým polem. V tomto případě tedy moc prostoru pro vylepšení výkonu nenacházíme.

Naopak vyhledání vhodného soupeře v *poli potenciálních soupeřů* poskytuje prostor pro zlepšení. Pokud existuje absolutní měřítko, které se používá pro ohodnocení páru, můžeme podle něj hráče seřadit a v takto seřazeném seznamu hráčů vyhledávat rychleji. V našem případě je ohodnocením páru *ohodnocení kvality jednotlivého zápasu*. Připomeňme, že to se počítá jako rozdíl Elo ohodnocení obou hráčů plus penalizace za nedávný zápas páru plus případná další ohodnocení páru.

Jak rozdíl Elo tak penalizace za nedávný zápas se jeví jako závislé hodnoty vztažené k danému páru. Zatímco u penalizace za nedávný zápas tomu tak skutečně je, za rozdílem Elo ohodnocení se skrývá hodnota Elo ohodnocení daného hráče, která je neměnná. Seřadíme-li tedy hráče podle jejich Elo ohodnocení, pohybem v takto *seřazeném poli potenciálních soupeřů* směrem od daného hráče poroste i rozdíl Elo ohodnocení potenciálního páru, který by vzniknul spárováním potenciálního soupeře s daným hráčem (obr. 3.3, 3.4).



Obr. 3.3 Příklad hráčů seřazených podle Elo ohodnocení



Obr. 3.4 Rozdíl Elo ohodnocení hráčů vzhledem k prostřednímu hráči (označeným zeleně) z obr. 3.3

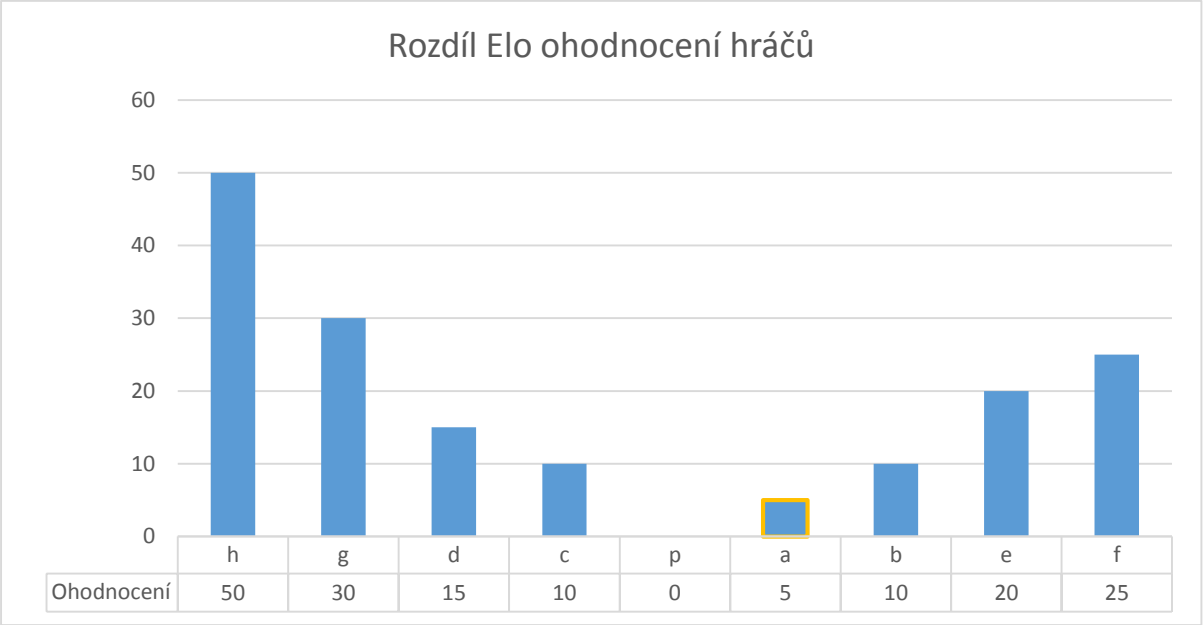
Tento mechanismus není plně přenositelný, lze ho však aplikovat na jakoukoli absolutní veličinu, která je zároveň *základem ohodnocení páru*. Pokud bychom u hráčů místo Elo ohodnocení sledovali například spolehlivost, tedy procentuálně vyjádřenou četnost příchodů na domluvené zápasy, mohli bychom aplikovat stejný způsob řazení a vyhledávání soupeřů na základě této veličiny.

Abychom mohli danou veličinu prohlásit za *základ ohodnocení páru*, je potřeba, aby ohodnocení páru bylo vždy větší nebo rovno této veličině. V našem případě, kdy má být rozdíl Elo ohodnocení hráčů *základem ohodnocení páru* to znamená, že penalizace za minulé zápasy ani žádné další přidané parametry, se kterými se pak rozdíl Elo ohodnocení hráčů sčítá, nesmí v součtu nabývat záporných hodnot. V případě, že by měla tato situace nastat, lze ji řešit posunutím rozsahu hodnot přičtením konstanty takové, aby nebyl součet těchto parametrů nikdy záporný. Protože se tato konstanta přičte při ohodnocení každého páru, nemá pak na provádění algoritmu ani jeho výsledek žádný vliv.

Při hledání soupeře se budeme postupně pohybovat na obě strany v *seřazeném poli potenciálních soupeřů* od daného hráče, pro kterého hledáme soupeře. Během provádění tedy roste rozdíl ohodnocení Elo mezi daným hráčem a jeho potenciálními soupeři. Uložíme si doposud nejvhodnějšího nalezeného potenciálního soupeře a k němu vztažené nejmenší nalezené *ohodnocení kvality jednotlivého zápasu*. To zpočátku nastavíme na maximální hodnotu datového typu, který jej reprezentuje. Při procházení *seřazeného pole potenciálních soupeřů* pak nejprve zjistíme, zda je rozdíl Elo ohodnocení daného hráče a nového potenciálního soupeře z pole menší nebo roven doposud nejmenšímu nalezenému *ohodnocení kvality jednotlivého zápasu*. Pokud tomu tak je, tento nový potenciální soupeř může být vhodnější než doposud nejvhodnější nalezený potenciální soupeř. Zkontrolujeme, zda spolu daný hráč a nový potenciální soupeř nebyli již spárováni a vypočteme jejich *ohodnocení kvality jednotlivého zápasu*. Pokud je toto ohodnocení menší než doposud nejmenší nalezené *ohodnocení kvality jednotlivého zápasu*, je tento nový potenciální soupeř označen jako doposud nejvhodnější nalezený soupeř.

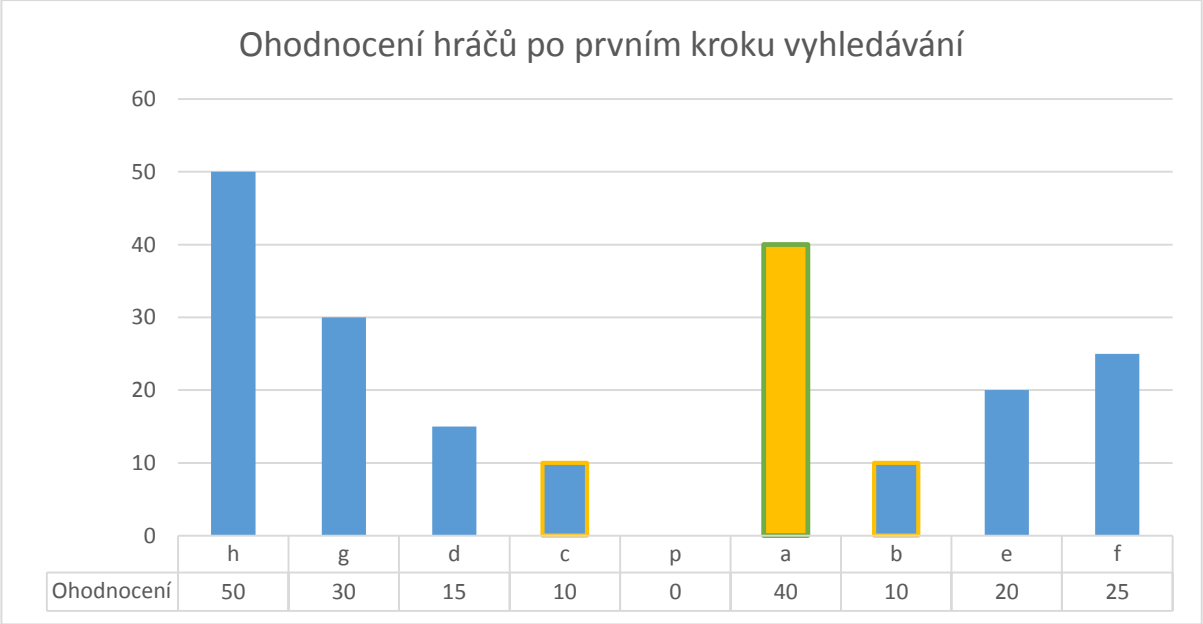
Pokud má daný hráč a potenciální soupeř rozdíl Elo ohodnocení vyšší, než je doposud nejmenší nalezené *ohodnocení kvality jednotlivého zápasu*, hledání soupeře na tuto stranu kolekce ukončíme. Protože platí, že rozdíl Elo ohodnocení dvou hráčů je menší nebo roven jejich výslednému *ohodnocení kvality jednotlivého zápasu*, je nemožné, aby byl v tomto případě potenciální soupeř kvalitnější než doposud nejkvalitnější nalezený soupeř.

Ilustrujme si nyní provádění vyhledávání na seřazených hráčích z obr. 3.3 a 3.4. Poznamenejme, že všechny uvedené hodnoty jsou smyšlené za účelem ilustrace průběhu vyhledávání bez nutnosti se zabývat dalšími dílčími detaily.



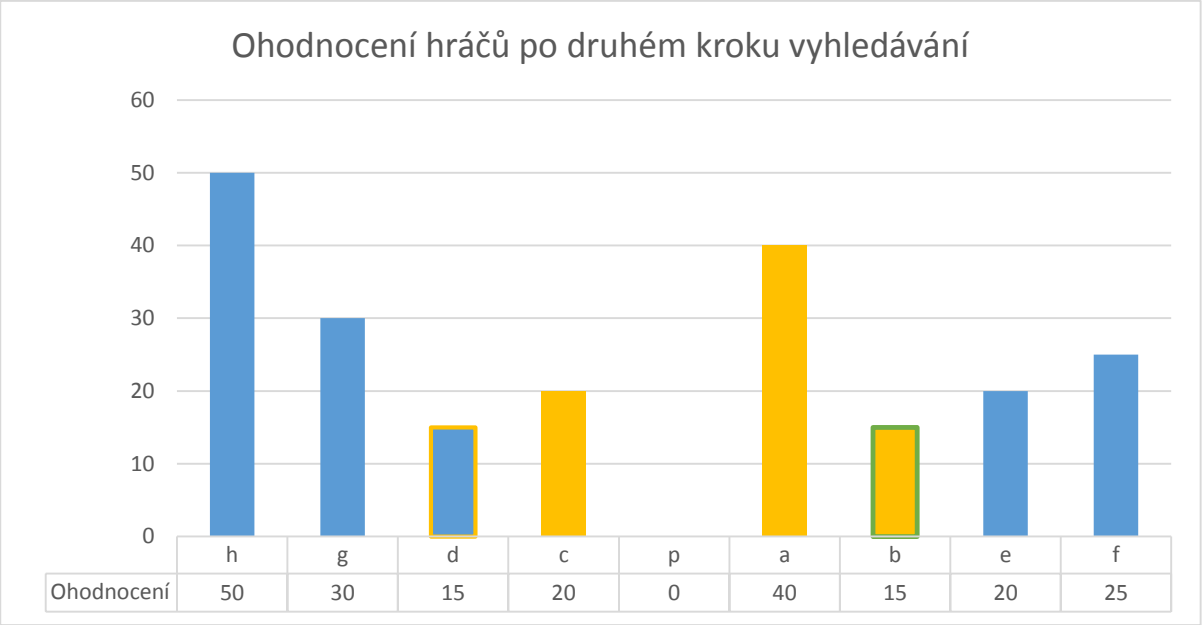
Obr. 3.5 Rozdíl Elo ohodnocení hráčů z obrázku 3.3 s vyznačeným prvním potenciálním soupeřem

Procházíme tedy potenciální soupeře hráče *p* (na obrázku uprostřed). Potenciální soupeř, který bude v dalším kroku ohodnocen, je označen žlutým ohraničením.



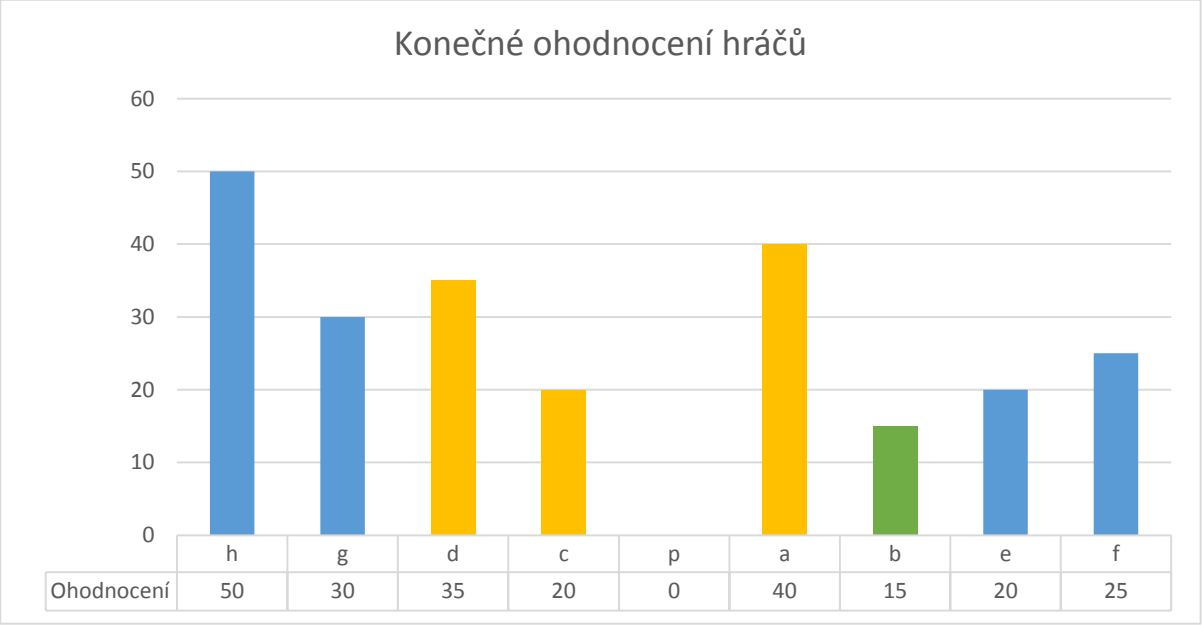
Obr. 3.6 Rozdíly Elo ohodnocení a skutečné ohodnocení hráče *a* vůči hráči *p* po prvním kroku vyhledávání

Po provedení prvního kroku známe skutečné *ohodnocení kvality jednotlivého zápasu* mezi hráči *p* a *a*. To je označeno žlutou barvou. Původní rozdíl Elo ohodnocení byl 5, skutečné ohodnocení ale vzrostlo v důsledku dalších parametrů tohoto ohodnocení (např. opakování zápasů z minulých kol). Hráč *a* se stává doposud nejvhodnějším nalezeným soupeřem s doposud nejnižším nalezeným ohodnocením 40. To označuje zelené ohraničení. V dalším kroku provedeme ohodnocení potenciálních soupeřů *b* a *c* vůči hráči *p*.



Obr. 3.7 Rozdíly Elo ohodnocení a skutečné ohodnocení soupeřů pro provedení druhého kroku vyhledávání

Po provedení dalšího kroku známe skutečné *ohodnocení kvality jednotlivého zápasu* hráče *p* vůči hráčům *b* a *c*. V případě hráče *b* je toto ohodnocení 15. To je doposud nejmenší nalezené ohodnocení, hráč *b* se tedy stává nejvhodnějším doposud nalezeným soupeřem. V následujícím kroku spočteme skutečné ohodnocení hráče *d*.



Obr. 3.8 Poslední krok vyhledávání

Ohodnocení kvality jednotlivého zápasu hráčů *p* a *d* jsme spočetli jako 35. Pár {*p*, *d*} je tedy méně vhodný než pár {*p*, *b*}, jehož ohodnocení je 15. V tuto chvíli prohledávání ukončíme, protože všichni doposud neprozkoumaní potenciální soupeři {*e*, *f*, *g*, *h*} mají rozdíl Elo ohodnocení vůči hráči *p* větší než doposud nejmenší nalezené ohodnocení 15, které odpovídá páru {*p*, *b*}. Protože platí, že *ohodnocení kvality jednotlivého zápasu* je větší nebo rovno rozdílu Elo ohodnocení hráčů, kteří se tohoto zápasu účastní, nemůže být už žádný z dalších potenciálních párů vhodnější než pár {*p*, *b*}.

Nalezli jsme tedy nejvhodnějšího soupeře *b* pro hráče *p* bez nutnosti procházet celé *seřazené pole potenciálních soupeřů*.

Ačkoli asymptotická nejhorší časová složitost vyhledání nejvhodnějšího soupeře zůstává na O(mn), je zjevné, že tímto přístupem vynecháme mnoho zbytečného procházení *pole potenciálních soupeřů*. Tím se vyhneme projití seznamu už vzniklých párů daného hráče a výpočet *ohodnocení kvality jednotlivého zápasu* pro každého z vynechaných soupeřů.

V případě, že budeme z tohoto pole navíc odstraňovat hráče, kteří již nemohou být spárováni, můžeme předpokládat, že nalezení nejvhodnějšího soupeře bude probíhat po celou dobu provádění algoritmu s velmi nízkým počtem prohledaných potenciálních soupeřů.

3.3 Dodatečné zpracování výsledku

Doposud jsme se zabývali návrhem párování a jeho co nejefektivnějšího provádění z hlediska asymptotické nejhorší časové a paměťové složitosti. Jak ale plyne z analýzy, zásadním nedostatkem výsledku získaného rozlosování je jeho nízké *celkové ohodnocení kvality zápasů*.

V důsledku toho, že hráči jsou párováni vždy s nejlepšími potenciálními soupeři, vznikají nejkvalitnější páry v prvních iteracích algoritmu. V posledních iteracích algoritmu tak v důsledku tohoto jevu vznikají páry s velmi nízkou kvalitou, kde většina nejvhodnějších protivníků daného hráče už má naplněn počet požadovaných zápasů, a tak nezbývá než jej spárovat s méně vhodnými soupeři.

Nízkou kvalitu rozlosování z hlediska *celkového ohodnocení kvality zápasů* lze řešit dodatečnou úpravou, zlepšením vzniklých párů. Protože *celkové ohodnocení kvality zápasů* je maximem ze všech dílčích *ohodnocení kvality jednotlivých zápasů*, můžeme jej snížit přímo snížením *ohodnocení kvality jednotlivého zápasu* nejméně kvalitního zápasu. V důsledku toho se ovšem s nejvyšší pravděpodobností stane to, že snížíme kvalitu jiného zápasu, protože potřebujeme někde vzít soupeře, které využijeme pro vytvoření nových párů. Měli bychom se vyhnout situacím, kdy bychom zlepšením kvality prvního páru snížili kvalitu druhého páru pod původní úroveň prvního páru. Navíc je potřeba zachovávat, nebo alespoň nějak zásadně nezhoršovat *celkové ohodnocení spárovaných hráčů* stávajícího rozlosování, tedy nerozpojovat páry s tím, že některé hráče necháme s nenaplněným počtem požadovaných zápasů.

Pro účely tohoto zpracování a zlepšování kvality budeme vždy rozpojovat původní páry a nahrazovat je novými tak, aby žádný ze vzniklých párů nebyl méně kvalitní než původní nejméně kvalitní pár. Přístupy se mohou lišit podle toho, do jaké hloubky umožníme rozpojování párů provádět. V případě nejmenší hloubky $h = 1$ jde o rozpojení dvou párů a prohození soupeřů v nich. S rostoucí hloubkou roste počet rozpojených párů r. Protože jeden pár k rozpojení vždy známe, platí pro počet rozpojených párů $r = h + 1$. Spolu s hloubkou roste výsledná kvalita a také časová a prostorová složitost tohoto přístupu.

Ilustrujme nyní toto navýšení. Pokud zvolíme metodu rozpojení vždy dvou párů, známe předem jeden z párů. Pro celkový počet n párů ve výsledném rozlosování pak vhodný pár nalezneme s asymptotickou nejhorší časovou složitostí O(n) a asymptotickou nejhorší paměťovou složitostí O(1).

Pokud zvolíme metodu rozpojení tří párů, tedy hloubku $h = 2$, situace se komplikuje. Známe sice původní pár, nicméně chybí nám další dva páry, které jsou na sobě závislé. To, že je daný pár nejvhodnější k částečné výměně s původním párem nemusí znamenat, že existuje třetí pár, jehož rozpojením a prohozením soupeřů s původním a daným párem vznikne kvalitnější rozlosování. Pro každý potenciální pár je tedy potřeba prohledat všechny potenciální páry s asymptotickou nejhorší časovou složitostí O(n), což vede na celkovou asymptotickou nejhorší časovou složitost O(n^2) pro n potenciálních párů. Asymptotická nejhorší paměťová složitost zůstává O(1). Uvědomme si ale, že oproti případu s prohozením dvou párů, kdy jsme si ukládali jen původní pár a hledali k němu nejvhodnější pár, nyní ukládáme jak původní pár, tak také druhý pár a hledáme pro ně třetí pár.

Obecně tedy platí, že pro přístup hledající páry do hloubky h je asymptotická nejhorší časová složitost O(n^h) a asymptotická nejhorší paměťová složitost O(h). Tato paměťová složitost plyne z toho,

že ukládáme vždy h průběžných potenciálních párů. Pro metodu rozpojení libovolného počtu párů s hloubkou $h = n$ pak dostáváme odpovídající asymptotickou nejhorší časovou složitost $O(n^h) = O(n^n) = O(n!)$ a asymptotickou nejhorší paměťovou složitost $O(h) = O(n)$.

3.3.1 Prohazování soupeřů mezi dvěma páry

Prohazování dvojic soupeřů mezi dvěma páry je nejjednodušším mechanismem prohazování soupeřů. V případě, že stávající rozlosování vykazuje vysokou nerovnoměrnost z hlediska *ohodnocení kvality jednotlivých zápasů*, je tento přístup efektivní a může kvalitu řešení znatelně zlepšit. S rostoucí rovnoměrností *ohodnocení kvality jednotlivých zápasů* daného rozlosování efektivita tohoto přístupu klesá. I při naprosté rovnoměrnosti však tento přístup může přinést zlepšení.

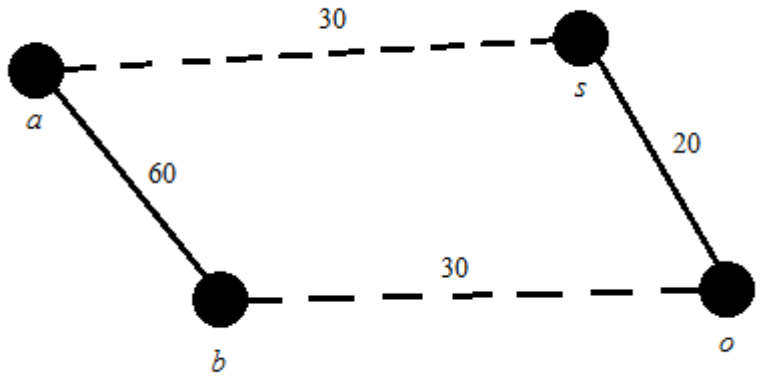
Protože je *celkové ohodnocení kvality zápasů*, které se snažíme zlepšovat, maximem ze všech dílčích *ohodnocení kvality jednotlivých zápasů*, budeme se vždy pokoušet zvýšit kvalitu nejméně kvalitního páru. Pro tento *maximální pár* platí, že má ze všech párů nejvyšší ohodnocení. V případě, že je *maximální pár* jen jeden, jeho rozpojením a spárováním původních soupeřů tak, aby každý z nově vzniklých párů měl ohodnocení nižší, než měl původní *maximální pár*, klesne také *celkové ohodnocení kvality zápasů*, což znamená zlepšení kvality řešení.

První ze dvou párů k prohození tedy známe. K nalezení druhého páru použijeme metodu, která je analogická k metodě výběru nejvhodnějšího soupeře popsané v předchozí podkapitole. Toto hledání nejvhodnějšího páru k prohození provedeme pro oba hráče z původního *maximálního páru*. V případě, že takto nalezneme dva různé páry vhodné k prohození, zvolíme ten, při jehož rozpojení a prohození soupeřů s bývalým *maximálním párem*, získáme dva páry, které budou mít menší maximální *ohodnocení kvality jednotlivého zápasu*.

Vyhledávání budeme provádět nad polem všech hráčů seřazených podle Elo ohodnocení. Pro hráče p z původního *maximálního páru* budeme procházet jeho okolí, dokud nenarazíme na obou stranách na soupeře s, jehož rozdíl Elo ohodnocení oproti hráči p je větší nebo roven původnímu ohodnocení *maximálního páru*. V takovém případě bychom vznikem páru $\{p, s\}$ nevytvořili lepší řešení, protože jeho *ohodnocení kvality jednotlivého zápasu* by bylo větší nebo rovno ohodnocení *maximálního páru*.

Pro *maximální pár* $\{a, b\}$ při prohledávání okolí hráče a v seřazeném poli všech hráčů vždy pro potenciálního soupeře s zjistíme, zda již pár $\{s, a\}$ neexistuje. Kdybychom v takovém případě pár $\{s, a\}$ vytvořili, porušili bychom základní požadavek na neopakování zápasů stejného páru v daném rozlosování. Následně projdeme všechny už vzniklé páry soupeře s a zjistíme, zda některý z jeho soupeřů není rovněž vhodným soupeřem pro hráče b. Musí platit, že takto vzniklý pár $\{b, o\}$, kde o je jedním ze soupeřů hráče s, ještě neexistuje, a zároveň, že *ohodnocení kvality jednotlivého zápasu* páru $\{b, o\}$ je menší než původní ohodnocení *maximálního páru* $\{a, b\}$ (obr. 3.9).

Následným vznikem párů $\{a, s\}$ a $\{b, o\}$ nejen, že nenarušíme kvalitu řešení z hlediska *celkového ohodnocení spárovaných hráčů*, ale rovněž tak snížíme počet *maximálních párů*. V případě, že *maximální pár* byl jen jeden, dojde přímo ke zlepšení *celkového ohodnocení kvality zápasů* tohoto řešení.



Obr. 3.9 Prohození soupeřů mezi páry $\{a, b\}$ a $\{s, o\}$ tak, že vzniknou nové páry (označeny přerušovanou čarou) s nižším maximálním ohodnocením. Povšimněme si rovněž nárůstu minimálního ohodnocení.

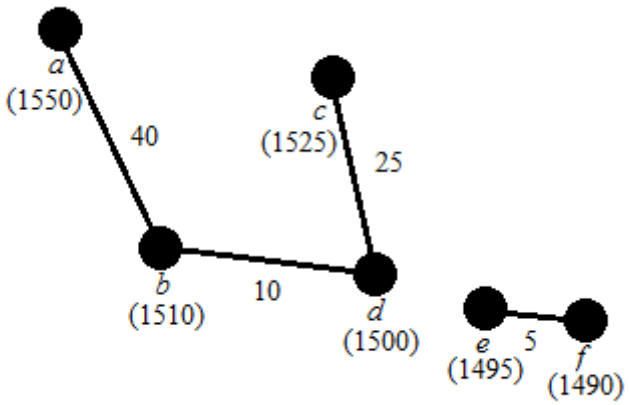
Může nastat situace, kdy daný pár je nejlepší možný, avšak jeho ohodnocení je vysoké. To se stává zejména u hráčů na okrajích rozložení Elo ohodnocení, kde je menší hustota hráčů, a logicky se tak budeme muset pro stejný počet naplněných požadovaných zápasů smířit s většími rozdíly Elo ohodnocení, než by tomu bylo blíže středu rozložení.

Přestože se nám už v takové situaci nepodaří zlepšit *celkové ohodnocení kvality zápasů*, mnoho *ohodnocení jednotlivých zápasů* ještě zlepšit jde. Pro porovnání takových rozlosování, kde tato situace nastává, je pak potřeba do *celkového ohodnocení kvality zápasů* zahrnout také vedlejší kritérium jako například průměr *ohodnocení kvality jednotlivých zápasů*. Z tohoto hlediska pak bude mít smysl provádět prohazování soupeřů mezi páry i v situacích, kdy prvním párem není *maximální pár*.

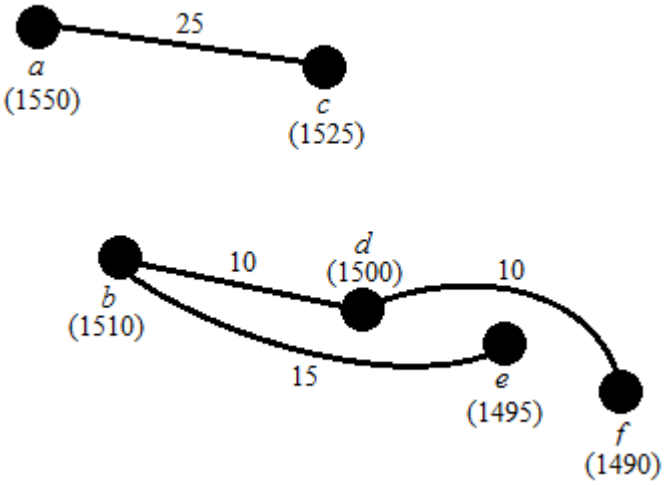
Protože *maximálních párů* může být v průběhu dodatečného zpracování stejný počet jako všech párů, zůstává asymptotická nejhorší časová složitost i v případě prohazování soupeřů všech dvojic párů stejná a to $O(n^2)$ pro n párů.

Dalším případem, kdy nelze hráče *maximálního páru* $\{a, b\}$ prohodit s vhodnými protikandidáty $\{c, d\}$, je situace, kdy je sice pár $\{a, c\}$ přípustný, ale pár $\{b, d\}$ již existuje. V takové situaci je nutné použít prohazování soupeřů s větší hloubkou, tedy že bychom vytvořili pár $\{a, c\}$ a pro hráče b a d bychom hledali další pár k prohození soupeřů. Za takových okolností musí pro úspěšné spárování existovat pár $\{e, f\}$, kde páry $\{b, e\}$ a $\{d, f\}$ ještě neexistují a jejich ohodnocení je menší než ohodnocení původního *maximálního páru*.

V případě, že nebudeme prohazovat dvojice jen pro *maximální pár*, ale pro všechny páry, by tato situace nastala pouze v případě, kdy by ohodnocení páru $\{e, f\}$ bylo menší než ohodnocení párů $\{b, e\}$ a $\{d, f\}$ (obr. 3.10 a 3.11). I přes tuto nízkou pravděpodobnost tohoto jevu ho však nemůžeme zanedbat jako příležitost ke zvyšování kvality výsledku.



Obr. 3.10 Příklad rozlosování, které nelze pomocí prohazování soupeřů ve dvojicích párů zlepšit. Ohodnocení hran představuje rozdíl Elo ohodnocení páru, čísla v závorkách pod názvy uzlů představují příklady odpovídajícího Elo ohodnocení hráčů.



Obr. 3.11 Optimální řešení příkladu z obrázku 3.10, kterého nelze dosáhnout prohazováním soupeřů mezi dvojicemi párů. Lze ho ale dosáhnout prohozením soupeřů mezi trojicemi párů.

3.3.2 Prohazování soupeřů mezi více páry

Prohazování soupeřů mezi dvěma páry tedy neřeší všechny přípustné situace. Na obrázcích 3.10 a 3.11 jsme ilustrovali příklad rozlosování, kde tento mechanismus selhává a je potřeba prohodit soupeře mezi více páry.

Při prohazování soupeřů mezi více páry můžeme využít prohazování soupeřů mezi dvěma páry. Nejprve zvolíme první počáteční pár. Pro ten vybereme nejvhodnější pár k prohození. Při prohození může zůstat nanejvýše jedna z dvojic hráčů rozpojena z důvodu, že z nich složený pár v tomto rozlosování už existuje a došlo by tedy k opakování zápasu stejného páru, nebo protože je ohodnocení této dvojice větší než původní ohodnocení počátečního páru.

Tuto nespárovanou dvojici zvolíme jako další počáteční pár a takto proceduru opakujeme, zanořujeme se, dokud nenastanou koncové podmínky. V případě úspěchu nezůstane žádná nespárovaná dvojice a použijeme takto vzniklé rozlosování. V opačném případě narazíme na situaci, kdy nelze hráče z počáteční dvojice prohodit s hráči z jiné dvojice a to buď z důvodu, že by to vedlo k opakování zápasu mezi stejnou dvojicí, nebo by takto vzniklý pár překročil ohodnocení prvního počátečního páru.

Oproti původnímu přístupu prohazování soupeřů mezi dvěma páry zde čelíme nutnosti upravovat rozlosování podle toho, jakým způsobem jsme ho v předchozích krocích ovlivnili, a zároveň zachovat původní rozlosování, abychom se k němu mohli vrátit v případě, že žádné z možných řešení není vhodné.

K tomuto problému můžeme zaujmout dvojí přístup. Jednak můžeme rozlosování při každém zanoření zduplikovat. Tak se dokážeme velmi snadno vrátit k předchozímu stavu. Problém ale je, že součástí rozlosování jsou rovněž seznamy existujících párů pro dané hráče, které je třeba aktualizovat. Pro n hráčů požadujících až m zápasů bychom museli zduplikovat n seznamů o délce $O(m)$, což představuje asymptotickou nejhorší časovou i paměťovou složitost $O(mn)$ pro každé zanoření. Pro h zanoření by to znamenalo asymptotickou nejhorší paměťovou a časovou složitost $O(hmn)$. Tento přístup by tak značně degradoval výkon a paměťovou spotřebu algoritmu a je tedy nevhodný.

Jiným přístupem by bylo ukládat pouze transakce reprezentující změny, které nastaly při zanoření na zásobník. Seznamy už spárovaných protivníků pro všechny hráče bychom tak zkopírovali jen jednou na počátku provádění dodatečného zpracování s asymptotickou nejhorší časovou a prostorovou složitostí $O(mn)$ pro n hráčů požadujících m zápasů. Během provádění bychom pak pracovali se zásobníkem změn o asymptotické nejhorší paměťové složitosti $O(h)$ pro h zanoření a seznamy spárovaných hráčů už jen průběžně upravovali podle těchto změn.

Pro n hráčů požadujících m zápasů pak při každém zanoření projdeme $O(mn)$ potenciálně vhodných párů a pro každý z $O(mn)$ vhodných párů se zanoříme. Provedeme tedy $O(mn)$ zanoření s celkovou asymptotickou nejhorší časovou složitostí $O((mn)^h)$, kde h je maximální počet zanoření. Asymptotická nejhorší paměťová složitost je při tom $O(mn + h)$.

V situaci libovolné hloubky zanoření $h = mn$ je tedy asymptotická nejhorší časová složitost $O(m!m^n n!n^m)$ a asymptotická nejhorší paměťová složitost $O(mn)$.

Tato čísla nepůsobí příliš příznivě, nicméně s přihlédnutím k provedeným optimalizacím při vyhledávání by i tento přístup mohl být na menší problémy použitelný. Jednak obvykle neprohledáváme celé pole potenciálních soupeřů. Dále můžeme předpokládat počet zanořování o něco nižší než $O(mn)$, protože už předem vynecháváme slepé větve, přes které bychom nezvýšili kvalitu stávajícího řešení. Dále lze předpokládat, že pro velké množiny hráčů ke spárování bude snazší najít páry k prohození a budeme se jen zřídkakdy zanořovat do maximální hloubky.

Navíc s přihlédnutím ke konkrétním vlastnostem řešeného problému, kde je počet hráčů ke spárování n řádově větší než počet požadovaných zápasů m, lze uvažovat $O(m) = O(1)$ a v důsledku tedy asymptotickou nejhorší časovou složitost $O(n!)$.

3.4 Architektura aplikace

Kromě samotného párovacího algoritmu a dalších algoritmů pro další zpracování výsledků musí výsledný program splňovat také další požadavky a umožňovat navíc tvorbu pomocných externích programů.

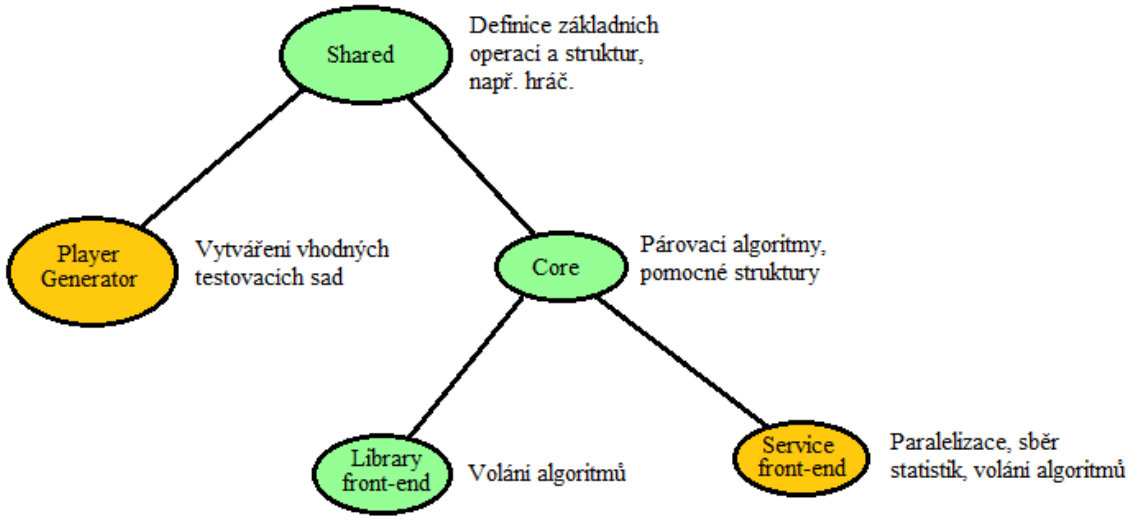
Za účelem efektivního vývoje a testování je potřeba, aby bylo možné program spustit jako aplikaci, kterou bude vývojář ovládat přes konzoli. V tomto případě je potřeba, aby aplikace uměla načítat konfigurace ze souborů, ukládat výsledky do souborů a provádět základní statistické operace a vyhodnocování nad výsledky. Mimo to jsou pro vývoj potřebné další doplňkové programy pro snazší manipulaci s daty uloženými v konfiguračních souborech a také pro generování vzorových testovacích dat.

Pro účely nasazení je pak potřeba, aby bylo možné aplikaci používat jak jako samostatně běžící službu, která bude dostupná po internetu, tak jako knihovnu, kterou bude možné zakomponovat do již existující webové služby, případně jiné aplikace.

Za těmito účely musí být jasně definováno rozhraní pro volání párovacích algoritmů. Jejich implementace tak bude v samostatné knihovně. V případě použití výsledného programu jako samostatné aplikace pak bude před tuto knihovnu postavena další vrstva. Ta bude podle typu aplikace buď zprostředkovávat síťovou komunikaci, nebo zpracovávání uživatelského vstupu z konzole a načítání konfigurací ze souborů.

Mimo to je ale pro pomocné nástroje potřeba vyčlenit další úroveň knihovny, kde se budou nacházet pouze základní abstrakce. Například nástroj k vytváření testovacích dat může využít stávající struktury reprezentující hráče a jejich historii zápasů ke generaci dat, které tak budou kompatibilní s daty používanými v jiných částech programu. Stejně tak i u dalších pomocných programů (např. pro vyhodnocování pokročilejších statistik) bude výhodné takto sdílet implementaci základních datových struktur.

Vzniká tak menší hierarchie modulů, kde kořenem je základní sdílená knihovna, kterou používá jak generátor hráčů a další pomocné nástroje, tak vyhodnocovací algoritmy. Na vyhodnocovacích algoritmech je pak závislý aplikační front-end, který by měl být nahraditelný, tj. pro vývoj a testování konzolová verze, pro výslednou aplikaci pak buď služba komunikující po síti, nebo knihovna.



Obr. 3.12 Hierarchie modulů: zelená – knihovny, žlutá – spustitelné soubory

Vzhledem k tomu, že je vyžadována velká míra nastavitelnosti algoritmů, je třeba při vyhodnocování posílat velký počet parametrů. Tyto parametry navíc vykazují hierarchické uspořádání. Pokud zvolíme jiný typ párovacího algoritmu, bude také potřeba posílat odlišné parametry. Z tohoto důvodu je výhodnější pro konfiguraci použít jeden kořenový objekt a pomocí návrhového vzoru skladba v něm umožnit předávání různých parametrů v dalších dílčích objektech, které podporují jednotné základní rozhraní. Tento konfigurační objekt bude předáván jako parametr volání algoritmu přes rozhraní knihovny.

4 Implementace

Implementačním jazykem knihoven a programů je C#, objektově-orientovaný programovací jazyk postavený nad rámcem .NET. Jako jazyk založený na jednoduchosti a dynamičnosti není zrovna nejvhodnějším kandidátem pro výkonově náročné aplikace. Byl vybrán primárně z důvodů snadné integrace do stávajícího systému Multiligy, který je postaven na rámci .NET, a jednoduchosti údržby ze strany administrátorů Multiligy.

4.1 Konfigurace a provedení algoritmu

Při každém použití párovacích algoritmů je třeba vědět, co vlastně přesně chceme. Který algoritmus použít, jaké nastavit váhy, jakým způsobem vybírat nejvhodnější hráče ke spárování a jejich protivníky.

K této konfiguraci slouží třída `Config`, která obsahuje některá základní nastavení společná pro všechny způsoby párování. Dále obsahuje odkazy na objekty, které specifikují typ algoritmu a jeho dílčí nastavení. Pro algoritmy založené na grafech je to například způsob generování grafu, zda má být vstup randomizován, popřípadě s jakou hodnotou inicializovat modul ke generaci náhodných čísel. Tak lze opakovat již proběhlý experiment se stejnými podmínkami pro účely ladění.

Před samotným vyhodnocením je pak podle této konfigurace zvolen příslušný algoritmus, je provedena inicializace (např. vytvoření grafu, instanciace tříd pro výpočty ohodnocení) a následně je algoritmus proveden. Podle dalších nastavení se pak během výpočtu mohou ukládat detailní data (jaké bylo ohodnocení páru, kolikátý je to pár pro oba hráče z dvojice). Algoritmus jako výsledek vrací množinu vzniklých párů a z důvodů ověření konzistence také seznam hráčů, u nichž nebyl naplněn počet požadovaných zápasů.

Po dokončení párování je provedena validace výsledku. Její kompilaci lze vypnout zrušením symbolu `VALIDATE_RESULT`. Validace ověří, že nebyl žádný hráč spárován sám se sebou, že nebyl vytvořen žádný duplicitní pár a že seznam hráčů, kteří nemají naplněn počet požadovaných zápasů, odpovídá výslednému rozlosování.

Výsledná data jsou pak z úrovně knihovny navrácena volajícímu. Ten nad nimi může provádět další operace, např. sběr statistik, náhodné procházení. Tohoto lze ve vrstvě aplikace určené k vývoji a testování využít k simulování použití algoritmu ke spárování několika kol nad stejnou sadou hráčů a sledovat tak také dlouhodobou kvalitu výsledků algoritmu.

4.1.1 Randomizace vstupu a výstupu

Vzhledem k tomu, že jsou všechny metody párování deterministické, dostaneme vždy pro stejnou vstupní sadu hráčů stejné rozlosování. Lze předpokládat, že v důsledku přibývajících kol do historie zápasů hráčů a změn v Elo ohodnoceních hráčů v důsledku proběhlých zápasů, se budou výsledky jednotlivých kol v praxi lišit.

Pro jistotu ale inicializace všech algoritmů nabízí možnost náhodného procházení vstupních dat. Tu můžeme použít i ke statistickému testování. Při volání algoritmu se stejným, ale náhodně seřazeným vstupem, můžeme zjistit, jakou mírou ovlivňuje uspořádání vstupu daný algoritmus.

Obdobnou operaci by bylo vhodné provést i u výsledného rozlosování, jinak je v závislosti na volbě algoritmu možné, že se znatelně projeví pořadí, v jakém byli hráči vyhodnocováni. Pak by mohlo jít vypozorovat například to, že první kolo hrají vždy hráči s nejvyšším počtem požadovaných zápasů s hráči požadujícími nejnižší počet zápasů. Je otázkou, zda je tento přístup pro uživatele skutečně

nevýhodný. Na Multilize, kde se hráči domlouvají, kdy spolu budou hrát, si můžeme představit situaci, kdy hráč požadující čtyři zápasy si domluví každý týden jeden zápas. Pokud bude postupně obvolávat soupeře a neuvědomí si, že by měl nejprve zavolat tomu, který požaduje jen jeden zápas, bude pak možná nucen přehodit některé zápasy, protože hráč s nízkým požadovaným počtem zápasů nebude mít s vyšší pravděpodobností v danou dobu čas.

4.2 Algoritmy

K volání algoritmů slouží statická třída `Algorithm`. Jejími parametry jsou seznam hráčů ke spárování a konfigurační objekt. Ten je předáván jako implementace rozhraní `IConfig` a je tedy možné místo něj použít libovolný vlastní objekt. Pro získání vzorových konfigurací slouží statická třída `Configuration`. Tyto vzorové konfigurace lze pak libovolně upravovat podle potřeb volajícího.

Párovací algoritmy jsou představovány objekty, protože obsahují vnitřní stav. To navíc umožňuje, aby algoritmy implementovaly jednotné rozhraní `IMatchingAlgorithm`, přes které probíhá jejich volání. Dále to umožňuje umístit inicializaci a konfiguraci jednotlivých typů algoritmů do oddělených tříd podle návrhového vzoru Továrna (angl. *Factory pattern*).

4.2.1 Výpočet ohodnocení

Součástí konfigurace párovacích algoritmů jsou objekty určené k výpočtu jednotlivých ohodnocení. Pro oba typy ohodnocení existují opět jednotná rozhraní, které implementují různé třídy. Ty jsou rovněž instanciovány za použití návrhového vzoru Továrna podle zadané konfigurace.

Do objektu algoritmu jsou pak při konstrukci dosazeny jako rozhraní s tím, že na nich algoritmus pouze volá požadované výpočetní funkce podle návrhového vzoru Strategie.

Pro výpočet *ohodnocení kvality jednotlivého zápasu* se používá rozhraní `IMatchEvaluator`, zatímco pro výpočet *ohodnocení počtu naplněných zápasů hráče* se používá `IPlayerEvaluator`. Objekty podporující tato rozhraní jsou vytvářeny přes příslušné Tovární třídy `MatchEvaluatorFactory` a `PlayerEvaluatorFactory`. V samotné aplikaci se nachází jedna až dvě implementace každého z těchto rozhraní. Jejich hlavním účelem je umožnit uživateli předat do algoritmu své vlastní objekty sloužící k těmto výpočtům. K tomu existují i potřebné konfigurační objekty, které umožňují předat algoritmu přímo objekt a ne jen nastavení, podle kterého se má vytvořit jeden z předdefinovaných objektů.

4.2.2 Grafový algoritmus používající haldu

Na základě myšlenky o grafové reprezentaci problému vzniknul algoritmus používající haldu. Protože je algoritmus velmi těsně propojen s typem používané haldy, je tento algoritmus reprezentován dvěma třídami `MatchingAlgorithmHeap` a `MatchingAlgorithmBucketHeap`. V prvním případě algoritmus používá jednoduchou binární haldu, v druhém případě algoritmus používá binární haldu, kde jsou prvky se stejnými klíči uloženy do polí, protože kolize klíčů se ukázala jako velmi častý jev.

V druhém případě je pak odebráním maxima z vrcholu haldy odebráno celé pole maximálních prvků. To umožňuje se vyhnout zbytečnému řazení, ale naopak komplikuje odebírání hráčů z haldy. Musíme vždy zkontrolovat, zda už daný hráč nebyl odebrán v rámci odebrání pole maximálních prvků.

Protože se přímá grafová reprezentace ukázala jako nevhodná z důvodu paměťové složitosti $O(n^2)$, umožňují tyto algoritmy také variantu, která vybírá soupeře, až když je to potřeba obdobným způsobem jako specializované algoritmy (popsáno v kapitole 3.2.2.2).

Vzhledem k tomu, že v takovém případě už se nejedná o grafový algoritmus, a že i samotné řazení pomocí haldy lze z hlediska výkonu překonat užitím řadících funkcí přímo v rámci .NET, další vývoj tohoto algoritmu pozbývá smyslu.

Jedná se sice o obecný přístup k řešení problému, nicméně vzhledem k výkonnostním potřebám je celý výsledný program laděn spíše konkrétně směrem k požadavkům systému Multiligy a tato obecnost se tak stává nevyužitelnou.

Pro n hráčů požadujících až m zápasů je asymptotická nejhorší paměťová složitost tohoto algoritmu $O(mn)$ až $O(n^2)$ podle zvoleného způsobu reprezentace dat. Algoritmus při svém vyhodnocování provede $O(mn)$ iterací, kde v každé iteraci odebere hráče z vrcholu haldy s asymptotickou nejhorší časovou složitostí $O(\log n)$, vyhledá k němu soupeře s asymptotickou nejhorší časovou složitostí $O(mn)$, vrátí hráče zpět do haldy s asymptotickou nejhorší časovou složitostí $O(\log n)$ a upraví ohodnocení soupeře s asymptotickou nejhorší časovou složitostí $O(n)$. Uložením současného indexu v poli haldy pro každého hráče lze asymptotickou nejhorší časovou složitost úpravy ohodnocení soupeře snížit na $O(\log n)$. Celková asymptotická nejhorší časová složitost je tedy $O(mn(3\log n + mn)) = O(m^2n^2)$. Pro Multiligu, kde platí $n \gg m$, lze předpokládat časovou složitost blížící se k $O(n^2)$ a paměťovou složitost $O(n)$, popř. $O(n^2)$ podle způsobu reprezentace dat.

4.2.3 Specializovaný algoritmus

Specializovaný algoritmus reprezentuje třída `MatchingAlgorithmSwap`. Ten byl už od první chvíle vytvářen se záměrem konkretizace úlohy a využití jejích aspektů k co nejefektivnějšímu provádění z hlediska časové složitosti.

Hráči ke spárování nejsou ukládání na haldě, nýbrž v poli, které je řazeno pomocí řadící metody rámce .NET. Ta je postavena na Introsortu [6], kombinaci Heapsortu, Quicksortu a Insertion sortu [7]. Lze předpokládat, že tato technika řazení bude efektivnější než halda, avšak asymptotická nejhorší časová složitost zůstává na $O(n \log n)$. Toto pole řadíme jen, pokud se mezi iteracemi změní *hladina ohodnocení* hráčů, čímž ušetříme zbytečné řazení hráčů.

V dalším poli jsou uloženi hráči seřazení podle Elo ohodnocení. Toto pole slouží k rychlému vyhledávání soupeřů tak, že prohledáváme soupeře s blízkým Elo ohodnocením a v případě rozdílu Elo ohodnocení většího, než má doposud nejvhodnější nalezený soupeř, hledání ukončíme. Detailní popis tohoto přístupu je uveden v podkapitole 3.2.2.2.

Jelikož se jedná o specializaci, je tento algoritmus vysoce efektivní a produkuje kvalitní výsledky. Na druhou stranu je pravděpodobně nepoužitelný na jakýkoli jiný problém, který se jen mírně liší od původního zadání.

Pro n hráčů požadujících až m zápasů je asymptotická nejhorší paměťová složitost $O(mn)$, protože u každého hráče je uložen seznam hráčů, se kterými byl již daný hráč spárován. K provedení algoritmu potřebujeme $O(mn)$ iterací, kde v každé iteraci s asymptotickou nejhorší časovou složitostí $O(1)$ odebereme hráče z konce seřazeného pole hráčů ke spárování a s asymptotickou nejhorší časovou složitostí $O(mn)$ vybereme nejvhodnějšího soupeře. Jednou za m iterací pak provedeme odebrání již uspokojených hráčů a řazení všech hráčů s asymptotickou nejhorší časovou složitostí $O(n \log n)$. Celková asymptotická nejhorší časová složitost algoritmu je tedy $O\left(mn\left(1 + mn + \frac{n \log n}{m}\right)\right) = O(m^2n^2 + n^2 \log n)$. Pro Multiligu, kde je počet požadovaných zápasů řádově menším než počet hráčů, je pak předpokládaná paměťová složitost $O(n)$ a časová složitost $O(n^2 \log n)$.

4.3 Dodatečné zpracování výsledku

Provedením algoritmu jsme sice získali rozlosování, nicméně zápasy, které podle něj proběhnou, budou vykazovat velké výkyvy v kvalitě. Abychom tyto výkyvy urovnali, je potřeba srovnat hladinu kvality zápasů. To znamená ty nejméně kvalitní zlepšit, ale naneštěstí často také ty nejkvalitnější zhoršit.

K tomu slouží dvě techniky prohazování soupeřů mezi páry popsaná v podkapitole 3.3. Implementována je zatím jen jednodušší varianta s prohazováním soupeřů mezi dvojicemi párů. Přesto je ale pro případ Multiligy, alespoň dočasně, i prohazování do neomezené hloubky výhodná varianta. V některých sportech na Multilize jsou poměrně nízké počty hráčů a v takových situacích nemusí jednoduché prohazování soupeřů mezi dvojicemi párů stačit k dostatečnému vylepšení kvality.

Samotné prohazování soupeřů mezi dvojicemi párů je v dosavadní implementaci poněkud nešťastně přímo zahrnuto ve specializovaném algoritmu, odkud také pochází jeho název `MatchingAlgorithmSwap`.

Do budoucna by bylo vhodné vydělit třídy pro prohazování soupeřů mezi páry. Všechny by implementovaly jednotné rozhraní, např. `IPairSwapModule`, a bylo by pak možné kombinovat libovolné algoritmy s libovolnými metodami dodatečného zpracování. To může být výhodné právě v situacích, kdy by na jeden sport bylo přihlášeno mnoho hráčů, kde bychom využili jen prohazování soupeřů v rámci dvojic párů. Na jiný sport by pak bylo přihlášeno menší množství hráčů a prohazování soupeřů mezi dvojicemi párů by nevedlo k dostatečně kvalitním zápasům. V takovém případě by přišlo na řadu prohazování soupeřů mezi páry do neomezené hloubky. Ta by v takovém případě byla relativně malá a tak by nedošlo k zásadnímu nárůstu spotřeby času během výpočtu.

4.3.1 Hráči s nenaplněným počtem požadovaných zápasů

Na konci kapitoly o analýze jsme usoudili, že v samotném jádru algoritmu opustíme optimalitu z hlediska *celkového ohodnocení spárovaných hráčů* a naopak si tím připravíme dobré podmínky pro zlepšování *celkového ohodnocení kvality zápasů*. Rozhodli jsme se, že nebudeme volit soupeře podle jeho *ohodnocení počtu naplněných zápasů* ale naopak podle *ohodnocení kvality jednotlivého zápasu* soupeře vůči hráči, pro kterého vybíráme soupeře.

Protože nejsme vždy schopni naplnit počty požadovaných zápasů všech hráčů, stává se, že některým hráčům není naplněn jejich počet požadovaných zápasů. V důsledku ztráty optimality z hlediska *celkového ohodnocení spárovaných hráčů* se může stát, že s nenaplněnými požadovanými zápasy skončí hráč, pro kterého to není nejvhodnější.

Vzhledem k tomu, že při dodatečném zpracování výsledků probíhá prohazování soupeřů mezi páry, je možné jednoduše prohodit páry tak, aby s nenaplněným počtem požadovaných zápasů zůstal hráč, u kterého je tento jev nejpříznivější. Pro výběr tohoto hráče pak můžeme specifikovat samostatnou strategii, která nebude kolizní s výběrem hráče ke spárování během provádění algoritmu.

K ohodnocení hráčů podle toho, zda by měli zůstat s nenaplněnými požadovanými zápasy, slouží třídy implementující rozhraní `IUnmatchedEvaluator`. Volba implementace a její vhodná inicializace je opět řešena přes konfigurační objekt `Config` a jeho dynamické složky, v tomto případě `IUnmatchedEvaluatorSettings`.

Po ohodnocení hráčů jsou vybráni ti, kteří mají zůstat s nenaplněným počtem požadovaných zápasů. Vždy, když je některému z těchto hráčů zrušen naplněný zápas, je přepočítáno jeho ohodnocení pro odebrání dalšího zápasu. Je ale potřeba zabránit tomu, aby byl daný hráč znovu spárován. Odebráním zápasu se totiž daný hráč může ocitnout ohodnocením níže než jiní hráči, kterým by byl v důsledku toho odebrán zápas a přidělen zpět danému hráči.

V případě, že je hráčů, kterým by měl být odebrán zápas, více se stejným ohodnocením k odebrání, je potřeba mezi nimi vybrat. Implementace zatím umožňuje buď vybrat náhodně, nebo vybrat prvního v seznamu. Do budoucna by bylo vhodné zde také zavést možnost detailnější konfigurace.

Takto získáváme možnost jednoznačně a přesně definovat, kteří hráči mají zůstat s nenaplněným počtem požadovaných zápasů. Zásadní výhodou tohoto přístupu je především jeho oddělenost. Není už potřeba při práci s *ohodnocením počtu naplněných zápasů hráče* vždy uvažovat také vliv na to, který hráč zůstane s nenaplněným počtem požadovaných zápasů.

5 Testování

Abychom mohli posoudit kvalitu vzniklých knihoven a programů k párování soupeřů, je třeba přistoupit k praktickým testům. Prvním cílem této bakalářské práce bylo vytvořit kvalitnější párovací algoritmus, než je algoritmus, který Multiliga původně používala. Druhým cílem bylo, aby byl tento nový párovací algoritmus škálovatelný. Aby byla jeho náročnost na výpočetní zdroje rostla přípustná i pro větší počty hráčů, než s jakými Multiliga v současnosti pracuje.

5.1 Srovnání kvality výsledků algoritmů

Pro porovnání kvality výsledků budeme používat rozlosování vytvořená třemi algoritmy: původní algoritmus Multiligy, nový grafový algoritmus a nový specializovaný algoritmus. Budeme sledovat, jak se zlepší aspekty kvality rozlosování v případě základní metody bez dodatečného zpracování, kterou reprezentuje grafový algoritmus, a v případě metody s dodatečným zpracováním, kterou reprezentuje specializovaný algoritmus.

Lze předpokládat, že mezi původním algoritmem a novým grafovým algoritmem bude značný rozdíl ve všech aspektech. Specializovaný algoritmus pak řeší spíše extrémní situace, kdy grafový algoritmus selhával, takže mezi specializovaným a grafovým algoritmem můžeme předpokládat spíše menší rozdíly v kvalitě.

algoritmus	**původní**	**grafový**	**specializovaný**
počet zápasů	34	35	35
počet hráčů s nenaplněným požadovaným počtem zápasů	2	1	1
detail nenaplněných zápasů*	2/4, 2/3	3/4	4/5
nejvyšší rozdíl Elo ohodnocení	187	101	74
nejnižší rozdíl Elo ohodnocení	1	1	2
průměrný rozdíl Elo ohodnocení	38.3	27.6	26.0
modus rozdílu Elo ohodnocení	20	8	8
opakované zápasy 1. k. / 2. k. / 3. k.	2/0/1	0/0/0	0/0/0

Tabulka 4.1 Porovnání kvality výsledného rozlosování mezi algoritmy.

*) Nenaplněné počty požadovaných zápasů jsou uvedeny ve formátu *naplněné*/*požadované* a pro jednotlivé hráče s nenaplněnými počty požadovaných zápasů odděleny čárkami.

Pro porovnání byl použit reálný vzorek 32 hráčů z Multiligy, konkrétně jde o hráče přihlášené do tenisové soutěže na duben 2015. Ze získaných dat vyplývá, že cíl neopakování zápasů se podařilo naplnit. Oproti původnímu algoritmu, který opakoval dva páry z předchozího kola a jeden pár z rozlosování tři kola zpět, u nových algoritmů žádné opakování nenastává, přestože se celkový počet zápasů podařilo zvýšit.

Za povšimnutí stojí také zlepšení kvality z hlediska hráčů s nenaplněným počtem požadovaných zápasů, kde oproti původní situaci, kde zůstal jeden hráč se dvěma nenaplněnými zápasy ze čtyř požadovaných a další hráč s jedním nenaplněným zápasem ze tří požadovaných, nové algoritmy ponechaly pouze jednoho hráče s jedním nenaplněným požadovaným zápasem. V případě bez dodatečného zpracování jde o hráče požadujícího čtyři zápasy, v případě s dodatečným zpracováním, jde o hráče požadujícího pět zápasů.

Mimo to došlo také ke zlepšení v ohledu kvality jednotlivých zápasů. Zatímco nejméně kvalitní zápas v původním rozlosování měl rozdíl ohodnocení Elo 187, který přibližně odpovídá poměru sil

75:25, v nových rozlosováních jsme dosáhli maximálního rozdílu ohodnocení Elo 101 a 74, kterým přibližně odpovídají poměry sil 64:36 a 60:40. Hodnota 74 je minimální, protože mezi dvěma nejlepšími hráči ve vzorku je právě tento rozdíl Elo ohodnocení. Jejich Elo ohodnocení jsou 1749 a 1675.

Oproti původnímu algoritmu došlo ke znatelnému zlepšení ve všech ohledech. Po dodatečném zpracování výsledku došlo k takřka nepocítitelnému zhoršení u nejkvalitnějších zápasů a naopak ke zlepšení u zápasů nejméně kvalitních. Navíc byl zvolen i vhodný hráč, který zůstal s nenaplněným počtem zápasů, díky možnosti přímo specifikovat strategii výběru tohoto hráče během dodatečného zpracování.

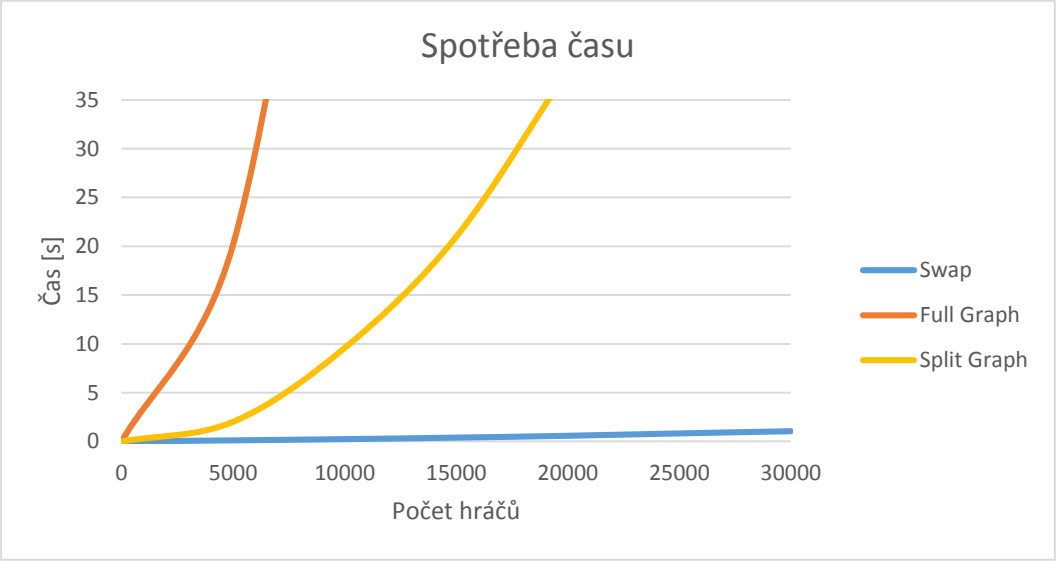
5.2 Srovnání výkonu nových algoritmů

Přestože jsme v předchozích kapitolách došli k určitým teoretickým závěrům ohledně asymptotické nejhorší časové a prostorové složitosti algoritmů, podívejme se nyní na to, jak výkonné jsou tyto algoritmy v praxi.

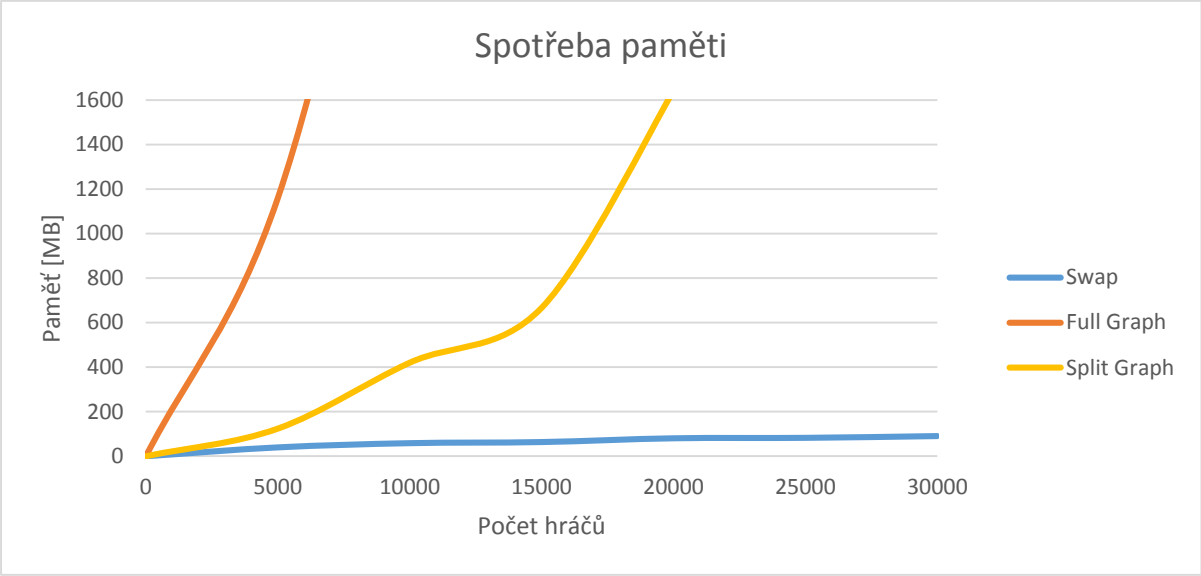
Grafový algoritmus s hráči reprezentovanými jako uzly grafu a potenciálními soupeři spojenými hranami, v grafech označený jako *Full Graph*, dopadl nejhůře. Jeho asymptotická nejhorší paměťová složitost $O(n^2)$ způsobuje rychlé zaplnění paměti a také neefektivní využívání pamětí cache procesoru. Navíc kvůli nutnosti rušit $O(n)$ hran pokaždé, když je naplněn požadovaný počet zápasů některého hráče, je jeho časová složitost značně degradována.

Grafový algoritmus *Split Graph*, který potenciální soupeře dohledává až na vyžádání a vytváří tak dynamické podgrafy, si stojí z hlediska paměti mnohem lépe. Kvůli dynamickému vytváření podgrafů předem neznámé velikosti ale stále vykazuje poměrně vysokou časovou složitost.

A konečně specializovaný algoritmus, v grafech označený jako *Swap*, vykazuje lineární paměťovou složitost. I přes předpokládanou asymptotickou nejhorší časovou složitost $O(n^2 \log n)$, která je horší než u ostatních algoritmů, obstál v testech nejlépe.

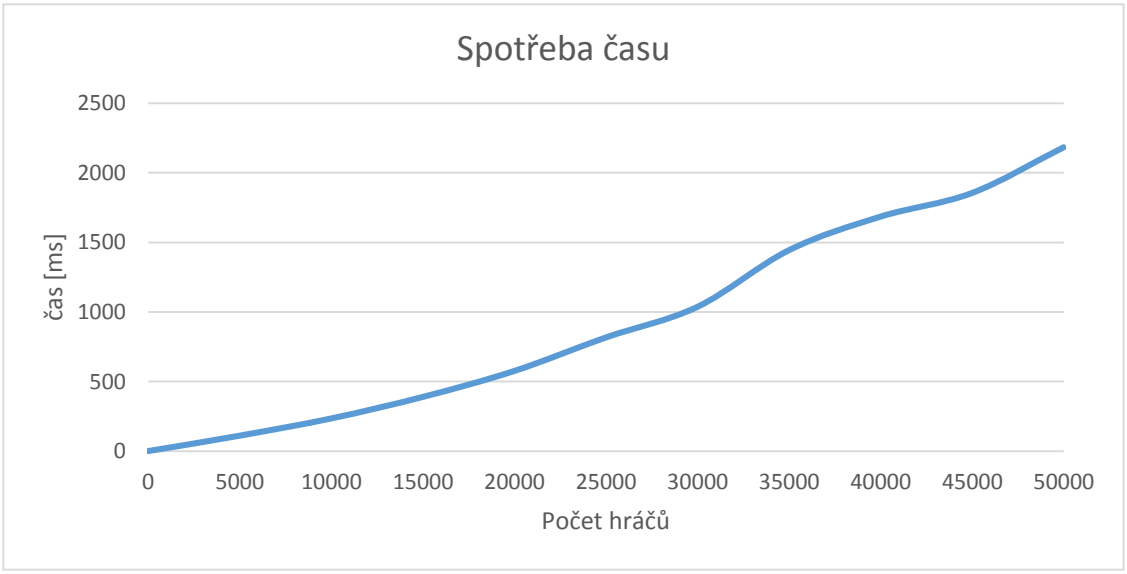


Obr. 4.1 Spotřeba času algoritmů



Obr. 4.2 Spotřeba paměti algoritmů

Nejefektivnějším algoritmem se tedy v praxi ukázal specializovaný algoritmus. Nejhorší asymptotická časová složitost specializovaného algoritmu je sice jednoznačně horší než u ostatních algoritmů, ale v praxi je tento algoritmus nejvýkonnější. Tento rozdíl můžeme přičíst tomu, že námi využívané optimalizace nemohly díky své nespolehlivosti snížit nejhorší asymptotickou časovou složitost specializovaného algoritmu. Jak se ale zdá, obvyklým scénářem jsou právě situace, kdy tyto optimalizační mechanismy fungují, zatímco jejich degradace na úroveň naivního přístupu nastává spíše vzácně.



Obr. 4.3 Spotřeba času specializovaného algoritmu

5.2.1 Vlastnosti testovacích dat

Algoritmy byly na výkon testovány na testovacích sadách s daným počtem hráčů. Tyto sady byly náhodně vygenerovány pomocí generátoru pseudo-náhodných čísel Mersenne Twister. Rozsah generovaného Elo ohodnocení hráčů je 1300-1700 a počet požadovaných zápasů hráče je 1-5. Rozdělení těchto náhodně generovaných veličin je uniformní.

Nad takto vygenerovanou sadou hráčů byly pak provedeny čtyři aplikace příslušného párovacího algoritmu za účelem vytvoření historie zápasů. Následně proběhlo deset (v případě dlouhého trvání výpočtu pouze pět) rozlosování bez úpravy historie zápasů. Pro každý počet hráčů bylo takto vytvořeno sto sad, nad kterými byl výkon testován. To znamená, že například pro 5000 hráčů bylo vytvořeno sto sad a v každé z těchto sad byly provedeny čtyři rozlosování k vytvoření historie zápasů a dalších deset rozlosování k měření spotřeby času a paměti. Celkem bylo tedy provedeno 100 * (4 + 10) = 1400 rozlosování, ze kterých bylo 1000 použito k vyvozování závěrů.

Prvním zjevným problémem je uniformní rozdělení Elo ohodnocení. Elo ohodnocení podléhá normálnímu rozdělení. Navíc by se se zvětšující testovací sadou mělo zvyšovat maximální a minimální ohodnocení, protože při větších populacích existuje vyšší pravděpodobnost na výskyt jedinců s mnohem lepšími schopnostmi.

Dalším problémem je rozdělení počtu požadovaných zápasů. To je rovněž uniformní, ovšem korektní rozdělení je zde velmi diskutabilní. Za účelem přesnějšího zjištění tohoto rozdělení by bylo potřeba statistická data pro tuto veličinu. Lze však předpokládat, že jednotlivců požadující vysoké hodnoty bude menší počet, protože zkoumaným systémem je amatérská ligy. Lze tedy předpokládat, že hráči obvykle nebudou chtít hrát velmi často.

Ač se tyto problémy mohou jevit jako překážka, existuje s vygenerovanými sadami jiný problém, který je zastíní. Testovací sady hráčů vznikají bez historie zápasů. Ta je dodána několika iteracemi algoritmu, po kterých se výsledné rozlosování zapíše do historie hráčů. V prvním takto vylosovaném kole tak vzniknou nejlepší možné páry s menší spotřebou z hlediska výpočetního času. V dalších kolech pak dochází k nutnosti hledat nejkvalitnější možné řešení mezi ne zcela ideálními párovacími příležitostmi a algoritmus pak počítá déle. A to zejména v případě aplikace dodatečného zpracování. Postupně pak vzniknou další zápasy a v situaci, kdy jsou ideální zápasy odstraněny z historie zápasů jako příliš dávná minulost, znovu se opakují tyto ideální zápasy spolu s poklesem spotřeby času.

Periodicky se tak opakují rozlosování, která jsou rychlá a mají velmi kvalitní výsledek. Tento problém vyřešíme tak, že při měření spotřebovaného času budeme vytvářet rozlosování bez jeho aplikace na historii zápasů. Vhodné je pak využít rozlosování zhruba uprostřed periody opakování ideálního rozlosování.

Při měření spotřebovaného času a paměti byl překážkou také rovněž samotný .NET rámec a operační systém Windows. Pro identický výpočet byla relativní chyba spotřeby času okolo 10%, což lze ještě považovat za přijatelné. Ovšem z hlediska spotřebované paměti narážíme na vnitřní mechanismy OS Windows při přiřazovaní paměti a zejména na nedeterminismus garbage collectoru. Typickým jevem je pak situace, kdy Windows zjistí, že program potřebuje o mnoho více paměti a tak mu jí přiřadí velké množství. V takové situaci se intenzita uvolňování paměti garbage collectorem sníží a k nárůstu dojde opět až při přiblížení se zaplnění tohoto paměťového bloku.

To se v případě našeho párovacího algoritmu může stát až po několika zvětšeních testovací sady. Nejprve dojde k razantnímu rozšíření paměti přiřazené procesu a garbage collector nechá nepoužitými daty zaplněnou větší část této paměti. Při zvětšování testovacích sad se bude rovněž stupňovat intenzita úklidu nepoužívaných dat, ale celková spotřebovaná paměť poroste jen pomalu. Až pak znovu u některé další testovací sady úplně dojde paměť přiřazená procesu. Operační systém pak procesu přiřadí další velký blok, takže dojde ke skokovému nárůstu spotřeby paměti. Tento jev lze pozorovat v obr. 4.2 u algoritmu *Split Graph*.

Data získaná na těchto testovacích sadách tedy nejsou vhodná pro utváření absolutně přesných závěrů o výkonnosti algoritmů. Jsou však dostačující pro porovnávání algoritmů mezi sebou, protože algoritmy řešily obdobné problémy, a i základní tvar funkcí vyjadřujících závislost na velikosti vstupu by se alespoň v případě časové složitosti neměl ve výsledku příliš lišit.

6 Závěr

V této bakalářské práci byl popsán algoritmus pro vytvoření a dodatečné zpracování rozlosování soupeřů pro kompetitivní sporty. Zabývali jsme se dvěma způsoby popisu párovací části algoritmu a dvěma způsoby dodatečných úprav, popsali jsme výhody a nevýhody každé z nich. Další popis se týkal samotného modulu, jehož součástí algoritmus je, včetně popisu okolních rozhraní a možností konfigurace za účelem snadného testování a porovnávání různých kombinací přístupů.

Takto popsaný modul byl implementován a úspěšně nasazen do produkčního fungování systému Multiligy. Dosavadní rozlosování se ke dni psaní této bakalářské práce zdají jako kvalitní a jsou každý měsíc od března 2015 používána k odehrání zápasů. Vzniklý modul podle zkušebních vyhodnocení naplňuje základní požadavky a dále nad nimi staví z hlediska kvality výsledku. Ve všech aspektech překonává svého předchůdce.

Z hlediska dalšího vývoje by bylo nejvhodnější nejprve provést refaktorování kódu. Zejména oddělit algoritmy od mechanismů dodatečného zpracování tak, aby bylo možné tyto části kombinovat nezávisle na sobě. Dále by bylo vhodné oddělit kód inicializující jednotlivé algoritmy od algoritmů samotných, protože je často do velké míry sdílený. Rovněž by bylo na místě vytvořit více modulů pro jednotlivé činnosti, které se v rámci algoritmů vykonávají. Takto by mohl vzniknout například modul k výběru nejvhodnějšího soupeře, který by se dal používat nezávisle na zvoleném algoritmu.

Mimo to by bylo vhodné implementovat dodatečné zpracování výsledku pomocí prohazování soupeřů ve větším počtu párů, které by na Multilize dobře posloužilo jak pro sporty s menším počtem hráčů, tak také například při rozšiřování systému do dalších měst, kdy lze zpočátku očekávat nízké počty hráčů v jednotlivých sportech.

Z obecných výsledků této práce uveďme například preferenci používání jednodušších datových struktur, nad kterými máme plnou kontrolu, a kolekcí, které jsou v paměti reprezentovány spojitě. Tím není myšleno, že bychom měli přestat používat složité struktury a věřit dostupným knihovnám. Naopak jde o zdůraznění platnosti myšlenky, že algoritmy a datové struktury jsou provázány jak mezi sebou [8], tak také s počítačovými prvky, na kterých fungují. Přestože máme dnes k dispozici mnohem výkonnější hardware než v době, kdy tato myšlenka vznikla.

Řešení je ve velké míře specializované. To je nezbytné pro použitelnost programu k řešení větších problémů. I přesto, že výsledný algoritmus má předpokládanou časovou složitost $O(n^2 \log n)$, k řešení problému čítajícího padesát tisíc prvků spotřebuje zhruba tři sekundy výpočetního času na jednom jádře procesoru, který bychom v době psaní práce mohli považovat za překonaný.

Obecnější varianta algoritmu dokázala na stejném procesoru za stejný čas vyřešit desetinásobně menší problém a její využitelnost je tak značně limitována. Pro zpracování padesáti tisíc prvků by obecný algoritmus potřeboval přibližně osm minut.

Literatura

[1] GASS, Saul I a Carl M HARRIS. 2000. *Encyclopedia of operations research and management science*. 2nd ed. Boston: Kluwer Academic, xxxviii, 917 p. ISBN 07-923-7827-X.

[2] KNUTH, Donald E. a Robert Endre TARJAN. 1976. Big Omicron and big Omega and big Theta. *ACM SIGACT News* [online]. **8**(2): 18-24 [cit. 2015-05-17]. DOI: 10.1145/1008328.1008329. ISSN 01635700. Dostupné z: http://portal.acm.org/citation.cfm?doid=1008328.1008329

[3] TARJAN, Robert Endre. 1985. Amortized Computational Complexity. *SIAM Journal on Algebraic Discrete Methods* [online]. **6**(2): 306-318 [cit. 2015-05-17]. DOI: 10.1137/0606031. ISSN 0196-5212. Dostupné z: http://epubs.siam.org/doi/abs/10.1137/0606031

[4] GLICKMAN, Mark E. 1995. A Comprehensive Guide to Chess Ratings. *Mark Glickman's World* [online]. [cit. 2015-01-17]. Dostupné z: http://www.glicko.net/research/acjpaper.pdf

[5] FREDMAN, Michael L. a Robert Endre TARJAN. 1985. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM* [online]. **34**(3): 596-615 [cit. 2015-05-17]. DOI: 10.1145/28869.28874. ISSN 00045411. Dostupné z: http://portal.acm.org/citation.cfm?doid=28869.28874

[6] Array.Sort Method. *Microsoft Developer Network* [online]. [cit. 2015-05-17]. Dostupné z: https://msdn.microsoft.com/en-us/library/kwx6zbd4(v=vs.110).aspx

[7] MUSSER, DAVID R. 1997. Introspective Sorting and Selection Algorithms. *Software: Practice and Experience* [online]. **27**(8): 983-993 [cit. 2015-05-17]. DOI: 10.1002/(sici)1097-024x(199708)27:8<983::aid-spe117>3.0.co;2-#.

[8] WIRTH, Niklaus. 1976. *Algorithms data structures=programs*. Englewood Cliffs, N.J.: Prentice-Hall, xvii, 366 p. ISBN 01-302-2418-9.

Příloha A

Obsah CD

Adresářová struktura CD

- `doc` – obsahuje manuály k spuštění a integraci programu.
- `res` – obsahuje vzorová nastavení algoritmu `default.cfg`, `bucket.cfg`, `full.cfg` a `mc.cfg`.
  - `res/in` – obsahuje pomocné skripty `GenerateAll.bat`, `Generator.bat` a `SubGenerator.bat`, které slouží ke generování vstupních sad hráčů.
- `src` – obsahuje zdrojové soubory a Visual Studio projekty.
  - `src/Shared` – obsahuje zdrojové kódy kořenové knihovny sdílené napříč všemi moduly.
  - `src/Core` – obsahuje zdrojové kódy algoritmů a konfigurace.
  - `src/Console` – obsahuje zdrojové kódy konzolového rozhraní k používání párovacího algoritmu.
  - `src/PlayerGenerator` – obsahuje zdrojové kódy programu ke generování testovacích sad hráčů.
  - `src/UnitTests` – obsahuje jednotkové testy pro složitější datové struktury.
  - `src/PerfTester` – obsahuje zdrojový kód nástroje k testování výkonu.