ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Generátor herního systému pro turnaje s rozdílnými výkonnostními kategoriemi
Student:	Michal Klement
Vedoucí:	Ing. Matěj Bartík
Studijní program:	Informatika
Studijní obor:	Teoretická informatika
Katedra:	Katedra teoretické informatiky
Platnost zadání:	Do konce zimního semestru 2018/19

Pokyny pro vypracování

Na základě slovního popisu (definující stavový prostor a požadavky na výsledné řešení) kombinatorického optimalizačního problému dodaného vedoucím práce formalizujte tento popis do formy vhodné pro řešení počítačem. Vedoucí práce zajistí vhodná testovací data.

Analyzujte a vyberte vhodnou metodu pro řešení tohoto pravděpodobně vícekriteriálního optimalizačního problému. Jako kandidáti na vhodné metody se jeví Integer Linear Programming, Constraint Satisfaction Problem nebo Pseudo-Boolean Optimization. Můžete však navrhnout další vhodné metody. Vezměte v potaz možnost on-line řešení.

Inspirovat se můžete současným (ale zastaralým) software TournaManage (viz Seznam odborné literatury).

Výstupem bakalářské práce budou podklady pro budoucí diplomovou práci, která se bude zabývat implementací a realizací "řešiče".

Seznam odborné literatury

http://www.tournamanage.net/currentrelease/
http://docs.tournamanage.net/doku.php
http://www.canoagem.org.br/arquivos/ckfinder/files/canoe%20polo%20rules%202015.pdf

doc. Ing. Jan Janoušek, Ph.D.
vedoucí katedry

prof. Ing. Pavel Tvrdík, CSc.
děkan

V Praze dne 24. března 2017

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra teoretickej informatiky

Bakalárska práca

Generátor herního systému pro turnaje s rozdílnými výkonnostními kategoriemi

Michal Klement

Vedúci práce: Ing. Matěj Bartík

16. mája 2017

Poďakovanie

Chcel by som poďakovať Ing. Matějovi Bartíkovi za vedenie práce, ochotu a ústretovosť a za všetky konzultácie, ďakujem Kateřine Boušovej za konzultáciu ohľadom programu TournaManage a veľmi by som chcel poďakovať svojej rodine a priateľke za dôveru a nekonečnú podporu.

Prehlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb., autorského zákona, v znení neskorších predpisov, a skutočnosť, že České vysoké učení technické v Praze má právo na uzavrenie licenčnej zmluvy o použití tejto práce ako školského diela podľa § 60 odst. 1 autorského zákona.

V Prahe 16. mája 2017 .....

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2017 Michal Klement. Všetky práva vyhradené.
Táto práca vznikla ako školské dielo na FIT ČVUT v Prahe. Práca je chránená medzinárodnými predpismi a zmluvami o autorskom práve a právach súvisiacich s autorským právom. Na jej využitie, s výnimkou bezplatných zákonných licencií, je nutný súhlas autora.

Odkaz na túto prácu

Klement, Michal. *Generátor herního systému pro turnaje s rozdílnými výkonnostními kategoriemi*. Bakalárska práca. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2017.

Abstrakt

Cieľom tejto práce je vypracovanie rešeršu a vytvorenie podkladov pre budúcu implementáciu softvéru na vytvorenie herného plánu pre turnaj v kanoepole. Rešerš je zameraný na možné metódy riešenia - problém s obmedzujúcimi podmienkami, lineárne programovanie a pseudo-booleovskú optimalizáciu, a takisto možnosť použitia on-line algoritmu. Z týchto možností som použil problém s obmedzujúcimi podmienkami pre formuláciu môjho problému, a vytvoril som databázový model, ktorý použijem pri implementácii. Výsledky tejto práce umožnia jednoduchšiu implementáciu.

Kľúčové slová problém s obmedzujúcimi podmienkami, lineárne programovanie, pseudo-booleovská optimalizácia, on-line algoritmus, kanoepolo, plánovanie, turnaj

Abstract

The aim of this work is to do a research and create a material for future implementation of software for creating canoepolo tournament schedule. The search is focused on possible methods of solution - constraint satisfaction problem, linear programming and pseudo-boolean optimization, as well as the possibility of using an on-line algorithm. From these options, I've used constraint satisfaction problem to formulate my problem and created the database model I will use in the application. The results of this work will help me with the implementation.

Keywords constraint satisfaction problem, linear programming, pseudo-boolean optimization, on-line algorithm, canoepolo, scheduling, round robin tournament

Obsah

Zoznam obrázkov

Zoznam tabuliek

Úvod

Pri sledovaní svojho obľúbeného športu málokomu napadne, čo všetko sa skrýva za organizáciou týchto podujatí. Samozrejme mám teraz na mysli návrh herného systému a určenie poradia zápasov, jednotlivých súperov a rozhodcov, a priradenie zápasov na ihriská. Tento problém môže byť triviálny, ale aj nabrať obrovských rozmerov. Všetko závisí od daných podmienok - počtu tímov, ihrísk, rozhodcov a ďalších doplňujúcich obmedzujúcich podmienok. Je jasné, že vymyslieť turnaj pre 4 alebo aj 8 tímov, ktorý sa musí odohrať za jeden deň na jednom alebo dvoch ihriskách a určiť jeden tím - ten najlepší, nie je vôbec ťažké. V kanoepole[1] sa však často hrajú turnaje celý víkend navyše v niekoľkých rôznych kategóriách. Väčšinou je dostupných viac ihrísk, takže sa hrá viac zápasov paralelne. Zápasu sa účastnia nie dva, ale rovno tri tímy, keďže rozhodcom je často jeden či viac hráčov z ďalšieho tímu. Takéto podobné podmienky vyplývajú z podstaty turnajov hry kanoepolo. Tie je potreba najprv formulovať, a potom vyriešiť jednou z dostupných metód. Táto práca je zameraná práve na analýzu týchto metód a formuláciu dôležitých podmienok pre turnaj v kanoepole.

Cieľ práce

Cieľom práce je vytvorenie podkladov, ktoré v ďalšej práci použijem k realizácii softvéru na generovanie herných systémov pre turnaje. Analýza je špecializovaná pre potreby kanoepola, avšak turnaje v tomto športe sa nelíšia tak výrazne od ostatných športov, aby softvér nemohol byť univerzálny a použitý aj pre iné účely.

Motivácia

V súčasnosti existujú veľmi obmedzené možnosti pre organizátorov turnajov v kanoepole. Používa sa jediný dostupný program TournaManage[2], ktorý je však pomerne zastaralý. Aj napriek tomu, že autor stále pracuje na aktualizáciách, veľa riešení je zbytočne komplikovaných. Namiesto toho, aby organizáciu turnajov uľahčoval, núti administrátorov vypĺňať celkom zbytočné a zdĺhavé formuláre. Používateľské rozhranie nie je výnimkou. Po konzultácii s ľuďmi, ktorí kanoepolo hrajú aj organizujú, som sa rozhodol, že im v rámci svojej bakalárskej a neskôr aj diplomovej práce pomôžem vytvoriť softvér presne podľa ich potrieb, aby som im uľahčil prípravy turnaja a umožnil maximalizovať športový zážitok.

Kapitola 1

Rešerš

Problém rozvrhovania vo všeobecnosti predstavuje veľmi širokú oblasť. Zaoberá sa ním veľa prác[3, 4, 5], a týka sa veľmi veľa odvetví. Mňa momentálne zaujíma oblasť športu. V tejto oblasti je pomerne známy a študovaný[6] *Traveling Tournament Problem*, kde je potrebné nájsť rozpis zápasov dlhodobých súťaží ako sú napríklad národné ligy v rôznych športoch. Pri tomto probléme je potrebné vyhovieť niekoľkým obmedzujúcim podmienkam. Jednou z nich je zákaz hrať dve hry po sebe vonku alebo doma. V tomto prípade je potrebné minimalizovať dĺžku ujdenej trasy. Navyše je často potrebné riešiť aj iné dôležité faktory ako dostupnosť štadiónov, alebo prianie majiteľov tímov ohľadne atraktívnych zápasov na konkrétne dni[7].

Môjho problému sa tieto obmedzenia netýkajú. Táto práca je zameraná na turnaje trvajúce jeden alebo dva dni na jednom mieste. Nejde o cestovanie ani domáce či vonkajšie zápasy. Dôležitý je férový rozpis zápasov, aby nedošlo k tomu, že jeden tím hrá 3 zápasy v rade a ďalší má dva zápasy prestávku. Tým by boli ovplyvnené ako aj výkony mužstiev, tak aj výsledky zápasov a turnaja. Tento problém sa môže zdať jednoduchý v prípade malého počtu tímov, avšak v kanoepole sa turnaje často organizujú naraz pre viac kategórií, takže tímov je omnoho viac. Väčšina klasických víkendových alebo niekoľko-dňových turnajov má pevne danú štruktúru, počet účastníkov aj herný systém, takže stačilo vymyslieť jeden univerzálny opakujúci sa rozpis. Naproti tomu majú turnaje v kanoepole veľa premenných. Jednotlivé podujatia sa môžu líšiť počtom kategórií, tímov a ihrísk, preto je potrebné vytvoriť herný plán pre každý turnaj zvlášť. V rešerši študujem práce, ktoré sa zaoberajú rozvrhovaním športových súťaží a zameriavam sa na použité metódy.

1.1 Traveling Tournament Problem

TTP je optimalizačný problém minimalizovania dĺžky cesty, ktorú každý tím precestuje počas sezóny. Tá pozostáva zo zápasov „každého proti každému“ dvakrát - doma aj vonku. Hlavným obmedzením je, že tím smie hrať maximálne U zápasov v rade doma alebo vonku. Vstupom je n, počet tímov, matica D rozmeru $n \times n$ so vzdialenosťami medzi jednotlivými mestami a parametry L a U, ktoré udávajú interval maximálnych možných zápasov doma či vonku v rade. Výstupom je potom turnaj s herným systémom „každý s každým“ dvakrát. TTP je zaujímavým problémom pretože kombinuje problém vyhovenia podmienok (zápasy doma/vonku v rade) a optimalizácie (minimalizovanie precestovanej vzdialenosti). Pre prvý typ problému dosahuje dobré výsledky programovanie s obmedzujúcimi podmienkami zatiaľ čo v druhom prípade je to celočíselné programovanie. Avšak ani jeden druh programovania si nevie poradiť s kombináciou vyššie uvedených problémov. Preto je potrebné nájsť kombináciu algoritmov[8]. Je dokázané[9], nielen že tento problém je *NP-ťažký* pre $U = 3$, ale aj po odstránení tohto obmedzenia zostáva problém *NP-ťažký*[10].

1.2 Problém s obmedzujúcimi podmienkami

Preložený termín sa veľmi často nepoužíva, zaužívaný je anglický termín *constraint satisfaction problem* a jeho skratka CSP. Keďže CSP všeobecne je *NP-ťažký* problém, neustále prebieha výskum techník a metód na hľadanie optimálnych riešení. V nasledujúcich riadkoch definujem CSP, a predstavím techniky a algoritmy, ktoré sa používajú.

1.2.1 Formálna definícia

CSP možno označiť ako množinu $P = (X, D, C)$, kde $X = \{x_1, ..., x_n\}$ je množina premenných, $D = \{D_1, ..., D_n\}$ množina domén pre každú premennú x_i a $C = \{c_1, ..., c_m\}$ množina obmedzujúcich podmienok, ktorým musí riešenie CSP vyhovovať. Doména D_i je množina všetkých možných hodnôt pre premennú x_i. Podmienka $C_i \in C$ je dvojica $< t_j, R_j >$, kde $t_j \subset X$ je podmnožina k premenných a R_j je k-násobný vzťah k odpovedajúcej podmnožine domén D_j. Riešenie CSP je priradenie každej premennej $x \in X$ hodnotu z odpovedajúcej domény tak, že vyhovuje všetkým podmienkam z C[11].

Všeobecne nás môže zaujímať akékoľvek riešenie, alebo môžme hľadať všetky riešenia, ktoré vyhovujú podmienkam, alebo požadujeme riešenie, ktoré je optimálne. V tom prípade existuje funkcia f definovaná pomocou niektorých, kľudne všetkých premenných z X, a snažíme sa nájsť také riešenie, ktoré má pre túto funkciu maximálnu, prípadne minimálnu hodnotu. Môže sa však stať, že pre daný CSP neexistuje ani jedno riešenie. Väčšinou kvôli príliš obmedzujúcim podmienkam, alebo príliš veľa podmienkam.

Sú najmä dva dôvody prečo použiť CSP[12]:

- Reprezentácia problému ako CSP veľmi pripomína pôvodný problém: premenné CSP priamo odpovedajú entitám problému, obmedzujúce podmienky nemusia byť vyjadrené ako nerovnice, čo pomáha problém lepšie formulovať a jednoduchšie pochopiť riešenie.
- Napriek tomu, že CSP algoritmy sú v podstate veľmi jednoduché dokážu niekedy nájsť riešenie rýchlejšie než lineárne programovanie.

1.2.2 Obmedzujúce podmienky

Sú esenciálnou súčasťou problému. Hľadané riešenie musí vyhovovať všetkým podmienkam. Podmienky sa môžu vzťahovať k 1 až n premenným. Ako *n-árnu* označujeme podmienku ak sa obmedzuje n premenných. Unárne podmienky vieme použiť už na predspracovanie problému. Ak máme napríklad podmienku, že otvárací zápas musí hrať domáci tím s tímom T, využijeme to a z domén hneď odstránime ostatné hodnoty.

Binárne podmienky sa vzťahujú k dvom premenným. Sú zaujímavé z hľadiska toho, že sa dajú znázorniť ako graf. Uzly predstavujú premenné, a hrana medzi nimi existuje v prípade vzájomnej obmedzujúcej podmienky.

Podmienky, ktoré sa týkajú viac než dvoch premenných nazývame globálne. Tie sa dajú rozložiť na niekoľko binárnych. To nám môže byť v mnohých prípadoch nápomocné, avšak nie vždy, pretože počet takto vzniknutých binárnych podmienok môže byť až exponenciálny. Príkladom najčastejšej globálnej podmienky používanej v CSP je *AllDifferent*, ktorá vraví, že každá premenná z množiny musí mať priradenú inú hodnotu.

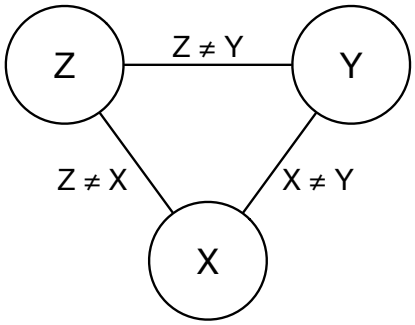
1.2.3 Binarizácia podmienok

Väčšina algoritmov pre CSP vyžaduje, aby obmedzujúce podmienky boli buď unárne alebo binárne. Preto je potrebné *n-árne (n>2)* podmienky upraviť na binárne. Potom je celý problém možné zobraziť ako graf. Uzly predstavujú premenné a hrany jednotlivé obmedzenia. Binárna podmienka je hrana medzi dvoma uzlami a unárna sa zobrazuje ako hrana, ktorá začína

aj končí v rovnakom uzle. Tieto cykly sa pre prehľadnosť z grafu odstraňujú, keďže unárne podmienky môžu byť okamžite vyhovené zredukovaním domén. Majme napríklad 3 obmedzenia:

- $X \neq Y$
- $Y \neq Z$
- $X \neq Z$

Tento problém môžeme zobraziť ako jendoduchý graf 1.1. Podmienka CSP

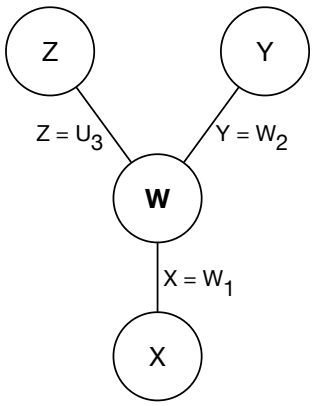


Obr. 1.1: Graf binárneho CSP

sa prekonvertuje na ekvivalentnú binárnu podmienku uvedením novej premennej, ktorá „obalí“ pôvodné. Takáto premenná sa nazýva *zapúzdrená* a jej doménou je Karteziánsky súčin domén jednotlivých premenných. Ak teda máme 3 premenné X, Y, Z a ich domény $D_X = \{1, 2\}$, $D_Y = \{3, 4\}$, $D_Z = \{5, 6\}$, môžeme vytvoriť zapúzdrenú premennú W s doménou $D_W = \{(1, 3, 5), (1, 3, 6), (1, 4, 5), (1, 4, 6), (2, 3, 5), (2, 3, 6), (2, 4, 5), (2, 4, 6)\}$. Ak pridáme podmienku $X + Y = Z$, doménu D_W môžeme ihneď zredukovať tak, aby vyhovovala tejto podmienke - $D_W = \{(1, 4, 5), (2, 3, 5), (2, 4, 6)\}$.

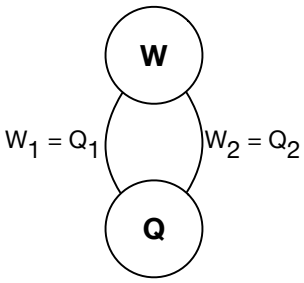
Novovytvorenú premennú však potrebujeme zakomponovať do pôvodného problému. To je možné dvoma spôsobmi:

- **S pôvodnými premennými**
  Skombinujú sa pôdovné premenné so zapúzdrenou. Vytvoria sa nové binárne obmedzenia pomocou jednoduchého vzťahu: $A = W_i$, kde A je pôvodná premenná, W je zapúzdrená premenná a i je pozícia A vo W.



Obr. 1.2: Graf binárneho CSP s pôvodnými premennými

- **Bez pôvodných premenných**
  Pridáme ďalšie obmedzenie $X < Y$ a následne tieto premenné zapúzdrime do Q s doménou $D_Q = \{(1,3),(1,4),(2,3),(2,4)\}$. Použijeme len zapúzdrené premenné a pridáme k nim binárne obmedzenia vzťahom $W_i = Q_j$, kde W a Q sú zapúzdrené premenné a i, j sú pozície v nich.



Obr. 1.3: Graf binárneho CSP bez pôvodných premenných

1.2.4 Hľadanie riešenia

Väčšina algoritmov na riešenie CSP systematicky prehľadáva množinu všetkých možných priradení hodnôt premenným. Výstupom takého algoritmu je buď nájdené riešenie alebo fakt, že riešenie neexistuje. Keďže premenných je obvykle veľké množstvo a domény obsahujú mnoho hodnôt, najväčšou slabinou týchto algoritmov je doba behu.

1.2.4.1 Generuj a testuj (GT)

Ako názov tejto metódy vraví, algoritmus najskôr vygeneruje nejaké priradenie a potom ho testuje. Prvé priradenie, ktoré vyhovuje všetkým podmienkam je riešenie. Veľkou nevýhodou tohto algoritmu je, že pokračuje v priraďovaní, aj napriek tomu, že nejaké obmedzenie je už porušené. Keďže ďalšie priradenia sa generujú čisto náhodne, veľmi ľahko sa môže stať, že algoritmus vyskúša väčšinu zo všetkých možností, čo trvá veľmi dlho.

1.2.4.2 Backtracking (BT)

Backtracking prináša vylepšenie oproti algoritmu *Generuj a testuj*. Funguje ako DFS, vyberie premennú a priradí jej nejakú hodnotu z jej domény ako GT, ale kontrola, či sa neporušila nejaká podmienka prebieha hneď. Ak áno musí sa vrátiť o krok späť a predchádzajúcej premennej skúsiť priradiť inú hodnotu a potom pokračovať ďalej až kým nenájde riešenie, alebo zistí, že riešenie neexistuje. Neustále kontrolovanie podmienok samozrejme nie je veľmi efektívne. Naivný BT totiž nemá prehľad o skutočnom dôvode problému, a teda sa nemusí hneď vrátiť do žiadaného miesta, a skúša veľa možností zbytočne.

1.2.5 Techniky pre udržanie súdržnosti

Súdržnosť(konzistencia) vyjadruje, že je všetko v „súlade“, teda priradenie hodnoty premennej je konzistentné ak neporušuje žiadnu podmienku. Tieto techniky sa využívajú na redukovanie stavového priestoru, aby bolo problém možné rýchlejšie vyriešiť.

1.2.5.1 Uzlová súdržnosť

Najslabší typ konzistencie. Uzol, reprezentujúci premennú X v grafe je uzlovo kozistentný, ak každá hodnota $x \in D_X$ spĺňa všetky obmedzujúce podmienky týkajúce sa X.

```
1: procedure NodeConsistency(csp)
   vstup: csp, problém s obmedzujúcimi podmienkami
2:     for each x v X do
3:         for each d v D_x do
4:             if nejaká unárna podmienka na x je nekonzistentná s d then
5:                 vymaž d z D_x
6:             end if
7:         end for
8:     end for
9: end procedure
```

1.2.5.2 Oblúková súdržnosť

Je najpoužívanejšia. Ide o 2-konzistenciu, ktorá môže byť použitá v pre-processingu aj ako súčasť algoritmu, ktorý udržiava konzistenciu. Jednoduchá definícia znie, ak máme obmedzenie C_{xy} medzi premennými X a Y, tak pre ľubovoľnú hodnotu z domény X existuje nejaká hodnota z domény Y tak, aby bolo obmedzenie C_{xy} splnené. To isté musí platiť aj opačne. V prípade použitia v pre-processingu oblúková konzistencia vylúči všetky hodnoty z domén, ktoré sú medzi sebou nekonzistentné. Poprípade sa tento algoritmus môže použiť po každom priradení hodnoty nejakej premennej.

Algoritmus 1 REVISE(x,y) vymaže z D_x, resp. D_y hodnoty, ktoré nie sú oblúkovo konzistentné. Vráti *true* ak vymazala nejakú hodnotu, inak *false*.

```
1: procedure ArcConsistency(csp)
   vstup: csp, problém s obmedzujúcimi podmienkami
2:     Q ← {C_xy v C, x ≠ y}
3:     repeat
4:         CHANGE ← false
5:         for each (x_i, x_j) v Q do
6:             CHANGE ← REVISE(x_i, x_j) alebo CHANGE
7:         end for
8:     until not CHANGE
9: end procedure
```

1.2.5.3 K-konzistencia

Definícia 1.2.1 *Nech $P = (X, D, C)$ je CSP.*

- *Majme množinu premenných $Y \subseteq X$ kde $|Y| = k - 1$, lokálna konzistencia I na Y je k-konzistentná práve vtedy keď pre akúkoľvek $k - tu$ premennú $x_{i_k} \in X \setminus Y$ existuje hodnota $v_{i_k} \in D(x_{i_k})$ taká, že $I \cap \{(x_{i_k}, v_{i_k})\}$ je lokálne konzistentná.*
- *CSP P je silno k-konzistentný práve vtedy keď je* j-konzistentný *pre všetky $j \leq k$.*

Uzlová konzistencia je 1-konzistencia a oblúková zas 2-konzistencia. K-konzistencia pre $k > 2$ sa môže použiť na redukciu domén napríklad pri spojení dvoch oblúkových konzistencií.

1.2.6 Šírenie podmienok

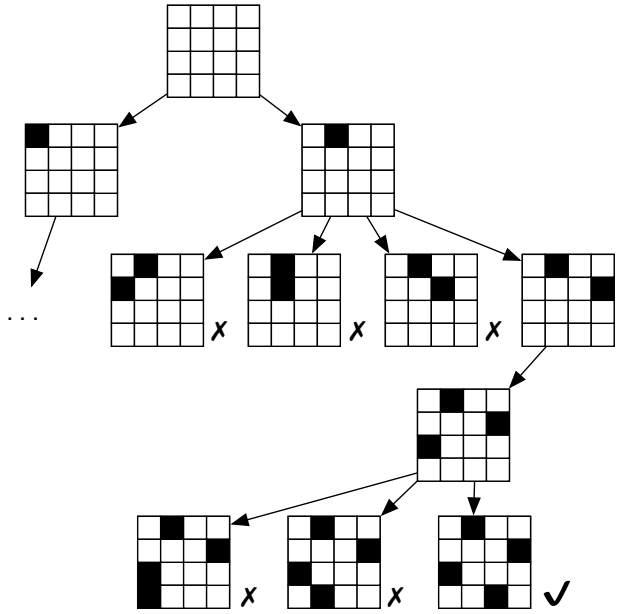
Backtracking a techniky udržania súdržnosti sú dva odlišné prístupy k hľadaniu riešenia CSP. Ďalší vznikne ak tieto dve spojíme a zakomponujeme algoritmus pre udržanie súdržnosti do základného backtrackingu. Ten klasicky rozširuje čiastočné riešenie tak, že vyberie ďalšiu premennú a priradí jej nejakú hodnotu. Po každom priradení sa potom použije nejaká z techník na udržanie konzistencie. V závislosti od použitého stupňa konzistencie dostávame rôzne algoritmy.

1.2.6.1 Backtracking

Základný BT si možno predstaviť ako kombináciu GT a 2-konzistencie. V priebehu hľadania riešenia testuje súdržnosť medzi premennými, ktorým už bola priradená hodnota. Algoritmus AC-Backtracking sa spustí vždy, keď je premennej x_n priradená nejaká hodnota. Nesúdržnosť detekuje ihneď ako sa objaví.

Algoritmus 2 REVISE(x,y) vymaže z D_x, resp. D_y hodnoty, ktoré nie sú oblúkovo konzistentné. Vráti *true* ak vymazala nejakú hodnotu, inak *false*.

1: **procedure** AC-Backtracking(csp, n)
   vstup: *csp*, problém s obmedzujúcimi podmienkami
   n, poradové číslo aktuálneho uzlu
2: $Q \leftarrow \{C_{in}$ v $C, i < n\}$
3: *consistent* $\leftarrow$ *true*
4: **while not** Q *prázdne* & *consistent* **do**
5: *vyber* a *vymaž* každý oblúk $(x_k x_m)$ z Q
6: *consistent* $\leftarrow$ REVISE(x_k, x_m)
7: **end while**
8: **return** *consistent*
9: **end procedure**



Obr. 1.4: Príklad *backtrackingu* pri probléme *4-dám* - čierne pole znamená umiestnenie kráľovny.

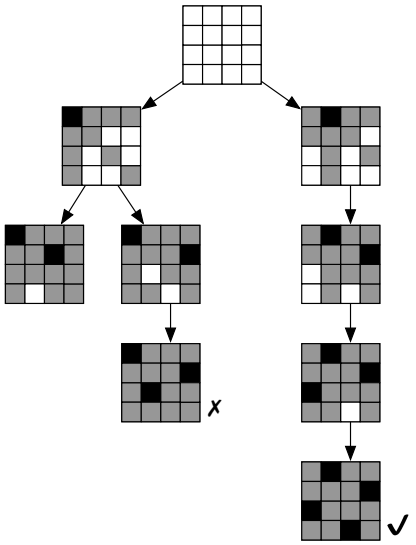
1.2.6.2 Forward checking

Aby algoritmus nemusel stále kontrolovať, či sú všetky podmienky po novom priradení splnené, používa sa tzv. *Forward checking*. Ide o techniku propagácie podmienok. Keď sa nejakej premennej priradí hodnota, z ďalších domén sa dočasne odstránia hodnoty, ktoré sú nekonzistentné s aktuálnym

priradením. Vďaka tomu algoritmus pozná, ktorá vetva stromu by viedla k neúspechu, a môže ju prerezať. V prípade, že sa z nejakej domény odstráni posledná hodnota, priraďovanie sa vráti o krok späť.

Algoritmus 3 REVISE(x,y) vymaže z D_x, resp. D_y hodnoty, ktoré nie sú oblúkovo konzistentné. Vráti *true* ak vymazala nejakú hodnotu, inak *false*.

```
1:  procedure AC-ForwardChecking(csp, n)
    vstup: csp, problém s obmedzujúcimi podmienkami
    n, poradové číslo aktuálneho uzlu
2:      Q ← {C_in v C, i > n}
3:      consistent ← true
4:      while not Q prázdne & consistent do
5:          vyber a vymaž každý oblúk (x_k x_m) z Q
6:          if REVISE(x_k, x_m) then
7:              consistent ← not D_k prázdne
8:          end if
9:      end while
10:     return consistent
11: end procedure
```



Obr. 1.5: Príklad *forward checkingu* pre problém *4-dám*. Čierne pole značí umiestnenie dámy, šedé pole je odstránené z domény, biele pole je voľné.

Pri porovnaní obrázkov 1.4 a 1.5 je vidieť, že *forward checking* dosahuje lepšie výsledky ako *backtracking*. Základny algoritmus sa dá aj pomerne je-

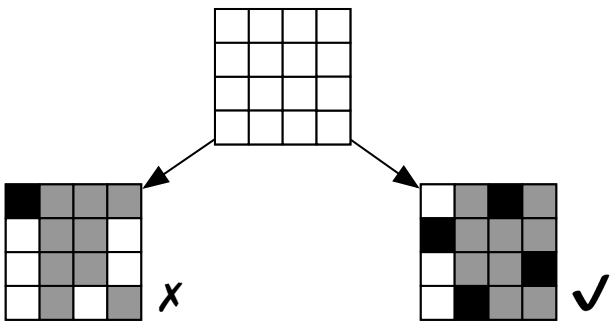
noducho vylepšiť[13]. Vďaka tomu sa zníži počet nekonzistencí, backtrackov a tým aj výpočtový čas.

1.2.6.3 Look Ahead

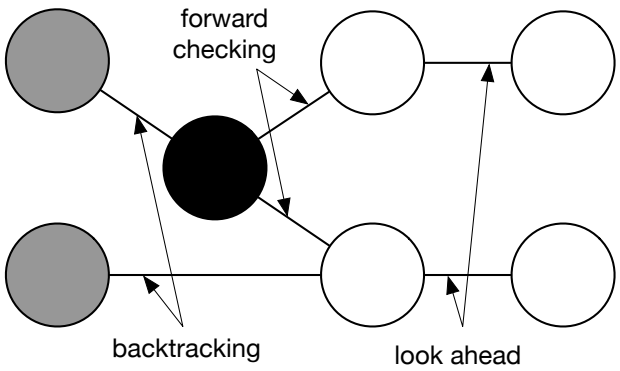
Algoritmus *look ahead* ide ďalej než *forward checking* a detekuje konflikty medzi premennými, ktoré budú v budúcnosti vyhodnocované. Vďaka tomu je možné vetvy vyhľadávacieho stromu, ktoré by viedli k neúspechu prerezať skôr než u predchádzajúceho algoritmu.

Algoritmus 4 REVISE(x,y) vymaže z D_x, resp. D_y hodnoty, ktoré nie sú oblúkovo konzistentné. Vráti *true* ak vymazala nejakú hodnotu, inak *false*.

```
1:  procedure AC-LookAhead(csp, n)
    vstup: csp, problém s obmedzujúcimi podmienkami
    n, poradové číslo aktuálneho uzlu
2:      Q ← {C_in v C, i > n}
3:      consistent ← true
4:      while not Q prázdne & consistent do
5:          vyber a vymaž každý oblúk (x_k x_m) z Q
6:          if REVISE(x_k, x_m) then
7:              Q ← Q ∪ {(x_i x_k) : (x_i x_k ∈ C, i ≠ k, i ≠ m, i > cv}
8:              consistent ← not D_k prázdne
9:          end if
10:     end while
11:     return consistent
12: end procedure
```



Obr. 1.6: Príklad *look ahead* pre problém *4-dám*. Čierne pole značí umiestnenie dámy, šedé pole je odstránené z domény, biele pole je voľné.



Obr. 1.7: Porovnanie jednotlivých techník propagovania podmienok. Šedé uzly majú hodnoty už priradené, čiernemu uzlu je hodnota aktuálne priraďovaná.

1.2.7 Techniky pre výber premenných a hodnôt

Vyhľadávacie algoritmy CSP vyžadujú definované poradie premenných, ktorým budú priraďovať hodnoty. Výber správneho poradia môže výrazne ovplyvniť efektivitu algoritmu.

1.2.7.1 Výber premenných

Existujú 2 základné princípy: *statické* poradie je dané ešte predtým než hľadanie začne a v priebehu sa nemení. Naopak *dynamické* vyberá ďalšiu premennú v priebehu hľadania na základe ďalších informácií. Z toho je jasné, že dynamické určovanie poradia nie je použiteľné pre základné vyhľadávacie techniky, ktoré nedisponujú žiadnymi dodatočnými informáciami o stave problému. Existuje niekoľko rôznych heuristík pre dynamický výber.

Najčastejšie sa používa *first-fail*, nazývaná aj *minimum remaining values (MRV)*. V tomto prípade sa vyberie premenná, ktorá má vo svojej doméne najmenej dostupných hodnôt. Touto metódou by sa mal zredukovať prehľadávaný strom, vďaka skoršiemu objaveniu nekonzisencie.

V prípade zhodnej veľkosti domén premenných sa použije iná heuristika. Tá vyberie premennú, ktorá sa vyskytuje v najväčšom počte obmedzení. Myšlienka je rovnaká ako v predchádzajúcom prípade.

1.2.7.2 Výber hodnôt

Okrem výberu premennej je dôležité vybrať hodnotu, ktorá sa ako prvá priradí vybranej premennej. To samozrejme neplatí v prípade, že nás zaujímajú

všetky možné riešenia. Ak ale hľadáme ľubovoľné riešenie, najrozumenšjie je vybrať takú hodnotu, ktorá obmedzí domény ostatných premenných čo najmenej. Inými slovami, nechá množiny možných hodnôt pre ďalšie voľné premenné, čo najväčšie.

1.2.8 Lokálne prehľadávanie

Na vyhľadanie riešenia je takisto možné použiť lokálne prehľadávanie. Uzly stromu stavového priestoru v tomto prípade predstavujú kompletné priradenie všetkým premenným, a jedna hrana znázorňuje zmenu hodnoty pri jednej premennej. Ľubovoľné priradenie samozrejme nebude konzistentné a tak je potrebné priradenia upravovať. Najpoužívanejšia heuristika u CSP je *Min-Conflicts*. Tá zabezpečí, že pri zmene hodnoty vybranej premennej sa použije hodnota, ktorá spôsobí najmenej nekonzistencí s ostatnými priradeniami.

Algoritmus 5 Algoritmus MIN-CONFLICTS[14] na riešenie CSP lokálnym prehľadávaním. Počiatočný stav môže byť vybratý náhodne alebo priraďovacím procesom, ktorý vyberie pre každú premennú hodnotu, ktorá spôsobí najmenej konfliktov. Funkcia CONFLICTS počíta počet obmedzujúcich podmienok, ktoré porušuje daná hodnota dosadená do aktuálneho priradenia.

1: **procedure** MIN-CONFLICTS(csp, max_steps)
   vstup: *csp*, problém s obmedzujúcimi podmienkami
   max_steps, maximálny počet krokov
   výstup: riešenie alebo neúspech
2: $current \leftarrow$ počiatočné kompletné priradenie pre *csp*
3: **for each** $i = 1$ do max_steps **do**
4: **if** *current* je riešenie pre *csp* **then return** *current*
5: **end if**
6: $var \leftarrow$ náhodne vybraná konfliktná premenná *csp*
7: $value \leftarrow$ hodnota v pre var, ktorá minimalizuje CONFLICTS($var, v, current, csp$)
8: nastav $var = value$ v *current*
9: **end for**
10: **return** $failure$
11: **end procedure**

Problém	Backtracking	BT+MRV	Forward Checking	FC+MRV	Min-Conflicts
USA	(> 1,000K)	(> 1,000K)	2K	60	64
n-Kráľovien	(> 40,000K)	13,500K	(> 40,000K)	817K	4K

Tabuľka 1.1: Problém *USA* je problém zafarbenia grafu, ktorý predstavuje 50 štátov USA, 4 farbami. Problém *n-Kráľovien* je klasická úloha rozmiestniť N kráľovien na šachovnicu rozmeru $N \times N$ tak, aby sa žiadne dve neohrozovali. Výsledok v tabuľke je pre všetky N od 2 do 50. *MRV* je heuristika *minimum remaining values*, čo znamená, že ako ďalšia premenná, ktorej sa bude priraďovať hodnota sa vyberie tá, ktorá má najmenej možných hodnôt - má najmenšiu doménu.

1.3 Pseudo-booleovská optimalizácia a lineárne programovanie

Pseudo-booleovská optimalizácia[15] a lineárne programovanie spolu veľmi úzko súvisia. Lineárne programovanie je metóda z oblasti optimalizácií a úlohy, ktoré rieši by mohli byť definované ako problémy maximalizácie alebo minimalizácie lineárnej funkcie $c^T x$ vzhľadom k lineárnym obmedzeniam $Ax \leq b$. x je vektor premenných, ktoré je potrebné určiť, c a b sú známe vektory koeficientov, A je známa matica koeficientov a $(\cdot)^T$ je operátor transponovania matice. V prípade, že $x \in \mathbb{Z}^n$ ide o *celočíselné lineárne programovanie*(ILP) a ak $x \in \{0,1\}^n$ jedná sa práve o optimalizáciu pseudo-booleovskej funkcie. Pre ukážku uvediem jednoduchý príklad[16]:

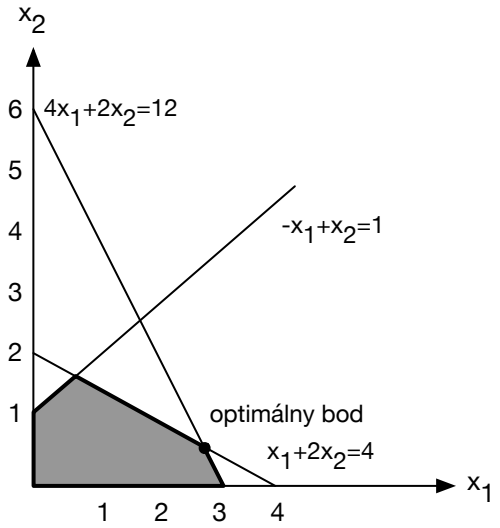
Nájdite čísla x_1 a x_2 také, aby súčet $x_1 + x_2$ bol najvyšší možný, aby platili podmienky $x_1 \geq 0$, $x_2 \geq 0$ a

$$x_1 + 2x_2 \leq 4$$

$$4x_1 + 2x_2 \leq 12$$

$$-x_1 + x_2 \leq 1$$

Prvé dve obmedzenia sú *podmienky nezápornosti*, ktoré sa používajú v podobných úlohách veľmi často. Ďalšie obmedzenia sú *hlavné obmedzenia* a *objektívna funkcia*, ktorú je treba v tomto prípade maximalizovať je $x_1 + x_2$. Keďže v tomto príklade sú len 2 premenné, riešenie je možné nájsť pomocou 2D grafu - plochy bodov, ktoré vyhovujú všetkým obmedzeniam. Túto plochu nájdeme ohraničením pomocou obmedzujúcich nerovníc. Tie rozdeľujú rovinu na dve polroviny. Hľadaná plocha je teda prienikom všetkých polrovín, ktoré vyhovujú podmienkam.



Obr. 1.8: Graf príkladu LP - šedá plocha je množina vyhovujúcich bodov.

Samozrejme nie všetky úlohy sa dajú vyriešiť takto jednoducho. Môžu obsahovať tisícky premenných a mnoho obmedzení. Z oblasti športu sa rozvrhovaním pomocou lineárneho programovania zaoberá napríklad práca[17] o kvalifikácii Južnej Ameriky na Majstrovstvá sveta vo futbale. Povaha môjho problému sa však od problému v spomínanej práci líši. Samotná formulácia nerovníc by spôsobovala zbytočné komplikácie narozdiel od formulácie obmedzujúcich podmienok pri CSP. Vyvodil som z toho záver, že môj problém bude ideálne formulovať práve ako CSP.

1.4 On-line riešenie

On-line algoritmus[18] prijíma sekvenciu požiadavkov a okamžite na ne reaguje. Každá sekvencia požiadaviek a reakcií na ne má určitú cenu. Najjednoduchším príkladom takého algoritmu je radiaci algoritmus *insertion sort*. V každej iterácii očakáva na vstupe jednu hodnotu, ktorú ihneď zaradí na správne miesto. Naproti tomu stojí napríklad *selection sort*. Ide o off-line algoritmus, má k dispozícii ihneď všetky prvky, ktoré následne zoradí do správneho poradia.

On-line algoritmy bývajú porovnávané s off-line algoritmami, ktoré dostanú celú sekvenciu vstupu naraz. Takisto je od nich vyžadovaná odpoveď na vstup s tým rozdielom, že je založená na celej sekvencii. Zjednodušene sa dá povedať, že off-line algoritmus pozná budúcnosť narozdiel od on-line

algoritmu. Je zrejmé, že neznalosť nasledujúceho vstupu predstavuje často nevýhodu. Preto mávajú on-line algoritmy o dosť horší výkon ako ich off-line verzie.

Nie každý on-line algoritmus má však aj off-line verziu. Existujú oblasti, kde je zdroj požiadaviek nepredvítateľný a je potrebné ihneď reagovať. Ako príklad môžem uviesť obchodovanie na burze. Požiadavku predstavuje cena komodity, na výber sú dostupné predať či kúpiť.

Určiť, čo je optimálny algoritmus pre off-line verziu je pomerne jednoduché. Taký algoritmus vyberie pre akúkoľvek sekvenciu vstupu sled akcií, ktoré majú minimálnu cenu. V prípade on-line algoritmov je to o niečo zložitejšie, pretože nech algoritmus vyberá akcie akokoľvek dobre, väčšinou príde vstup, ktorý spôsobí, že výsledok nebude ani zdaleka optimálny.

Preto bol definovaný *konkurenčný pomer*[19]. Ten udáva do akej miery je on-line verzia schopná „konkurovať“ svojej off-line verzii, teda o koľko je horšia. Konkurenčný pomer je definovaný ako maximálny pomer ceny akcií on-line algoritmu a optimálnej off-line verzie spomedzi všetkých vstupov. Optimálny on-line algoritmus je teda ten, ktorý má tento pomer najmenší.

V mojom prípade je známy celý nadchádzajúci vstup a teda nie je vyžadovaný on-line algoritmus. Teoreticky by bolo možné daný problém spracovať takýmto algoritmom. Podľa môjho názoru by to však nebolo výťažné, keďže, ako som už uviedol vyššie, on-line algoritmy majú znateľne horší výkon. Túto úvahu som potvrdil v mojom rešerši, kde som nenarazil ani na jednu prácu, ktorá by sa zaoberala on-line riešením plánovania turnaja, alebo aspoň podobnej problematiky.

1.5 Herné systémy

Vo svete športu existuje, a používa sa niekoľko rôznych herných systémov. Niektoré sú určené pre konkrétny počet tímov, ostatné sú univerzálne a líšia sa najmä celkovým počtom odohratých zápasov. V nasledujúcich odstavcoch zanalyzujem možné systémy vhodné pre môj prípad - turnaj v kanoepole.

1.5.1 Každý s každým

V angličtine označovaný termínom *round-robin tournament*(RRT) u nás všeobecne známy samopopisujúcim termínom „každý s každým“. Asi najznámenjší a najrozšírenejší systém naprieč všetkými športami. Všeobecne sa využíva v ligových súťažiach a skupinových fázach turnajov. Najviac sa hodí pre párny počet tímov, ale funguje aj v opačnom prípade. Základný

systém môže mať niekoľko variánt. Tie sa delia na základe počtu medzi sebou odohratých zápasov. Najčastejšie je to jeden raz - *single round-robin tournament*(SRRT) alebo dvakrát *double round-robin tournament*(DRRT). Ojedinele to môže byť viac kôl.

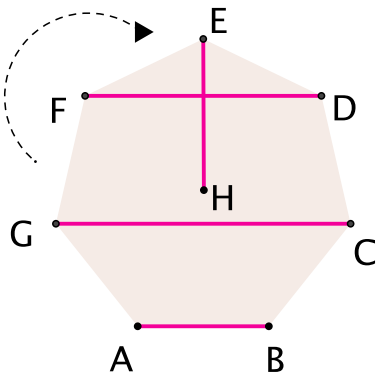
Ak označím počet kôl k a počet tímov n, v tomto systéme každý tím odohrá $k(n-1)$ zápasov, takže celkovo je to $k(n-1)\frac{n}{2}$. V prípade typického víkendového turnaja v kanoepole je žiadúce, aby sa prvá fáza turnaja odohrala v prvý deň. Preto by som tento systém zvolil ak by súčet zápasov vo všetkých skupinách všetkých kategórií vynásobený 30 minútami, čo je čas vymedzený na jeden zápas, bol nanajvýš rovný súčtu dostupného hracieho času na všetkých ihriskách. V inom prípade by to samozrejme nebolo možné. Pozitívom je, že aj v prípade malého počtu tímov je možné upraviť počet kôl, a tím aj počet zápasov pre jednotlivé tímy. Pre kanoepolo je totiž dôležité, aby každý tím odohral aspoň 5 zápasov za víkend. Navyše aj rozvrhovanie jednotlivých zápasov pre tento systém je jednoduché.

Už samotný názov musí pripomínať určité prvky z teoretickej informatiky, konkrétne teórie grafov. Mám na mysli úplný graf. Úplný neorientovaný graf s počtom uzlov n má $n\frac{n-1}{2}$ hrán. V mojom prípade teda uzly predstavujú tímy, a hrana medzi uzlami A a B predstavuje zápas medzi tímami A a B. Pri rozvrhovaní zápasov sa dá využiť vlastnosť, že ľubovoľný k-úplný graf, kde k je párne číslo, je 1-faktorizovateľný. To znamená, že je možné nájsť $k-1$ disjunktných 1-faktorov, čo odpovedá hranovému ofarbeniu. Každý 1-faktor potom znázorňuje jedno kolo turnaja. Problém nepárneho počtu tímov je možné jednoducho vyriešiť pridaním jedného triviálneho uzla. Tím, ktorý by mal v danom kole hrať proti tímu tohto triviálneho uzla bude mať prestávku.

Celý problém rozvrhovania zápasov je však možné vyriešiť aj veľmi jednoducho a elegantne pomocou metódy, ktorú vo svojej práci predstavil Édouard Lucas[20]. Pomocou grafu s $k-1$ uzlami po obvode a jedným v strede s vodorovnými hranami a jednou zvislou stačí uzly posúvať po obvode a postupne dostávame zápasy jedného kola.

1.5.2 Vyraďovací systém

Kvôli grafickému znázorneniu sa mu ľudovo hovorí „pavúk“. Veľmi populárny herný systém takmer vo všetkých športoch najmä v pokročilejšej fáze turnajov, kde sa rozhoduje o konečnom umiestnení tímov. Je použiteľný len na párny počet tímov. Jedna porážka v tomto systéme znamená pre tím koniec v turnaji, takže v každom kole postúpi polovica družstiev. Navyše to znamená, že počet tímov musí byť mocninou 2. Preto často prvá časť turnajov slúži na redukciu tímov na najlepších 2^N. Celkový počet zápasov



Obr. 1.9: Pomocou tejto grafickej pomôcky je možné rýchlo a jednoducho určiť dvojice súperov v jednotlivých kolách turnaja „každého s každým“. Uzly spojené hranou hrajú proti sebe, pre ďalšie kolo stačí posunúť uzly v smere hodinových ručičiek.

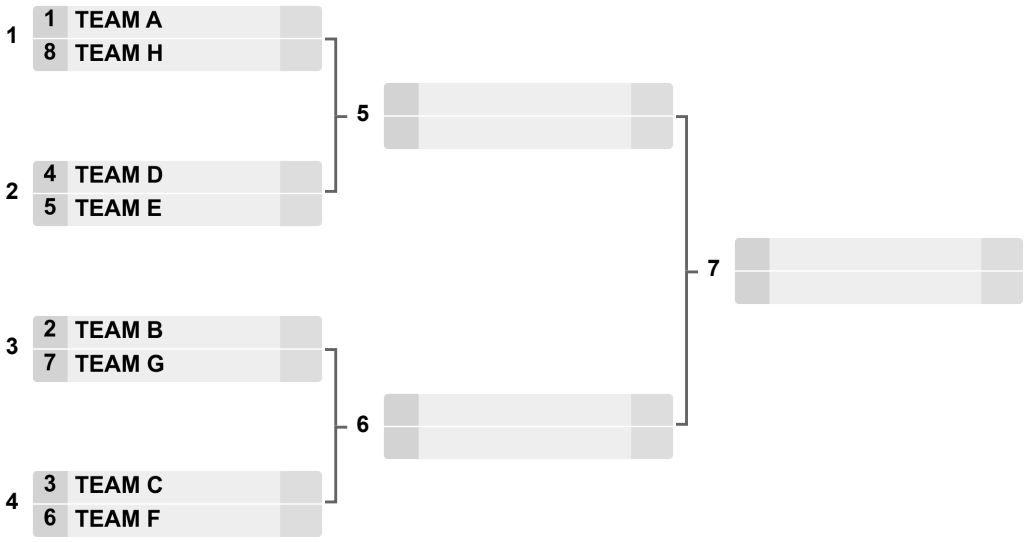
je $N - 1$, pričom finalisti odohrajú logicky $log_2(N)$ zápasov. Najčastejšie hrajú tímy v každom kole proti sebe len jeden zápas, ale je možné namiesto jedného zápasu hrať sériu zápasov, kde postupujúci tím bude ten, ktorý vyhrá vo viacerých zápasoch série. Môže sa tak hrať na 2, 3 či viac víťazných zápasov. Označuje sa to ako *BO3* (best of 3; najlepší z 3 zápasov) a podobne.

Nasadzovanie tímov do zápasov je najčastejšie riešené nasledovne. V 1. kole proti sebe hrajú tímy ak súčet ich seedu je $N+1$. To znamená prvý s posledným, druhý s predposledným tímom a tak ďalej. Tiež sa strieda umiestnenie zápasu v „pavúku“ tak, aby sa čo najviac oddialil prípadný možný stret dvoch najvyššie nasadených tímov. V opačnom prípade by prehra vo finále nemusela nutne znamenať 2. miesto. Zápas o 3. miesto je väčšinou voliteľný, a záleží na rozhodnutí organizátorov, či je dôležité určiť jeden konkrétny tím, ktorý skončí na 3. mieste.

Problémom pre turnaj v kanoepole je, že v dôsledku toho, že v každom kole vypadne polovica tímov, majú tímy rôzny počet odohratých zápasov. Pre slabšie tímy by to bolo nefér a preto je vhodné použiť vyraďovací systém až pre posledné 4 maximálne 8 tímov.

Existujú takisto rôzne varianty vyraďovacieho systému, ktoré môžu zaktraktívniť súťaž.

- Vyraďovací systém na dve porážky
- McIntyrov systém



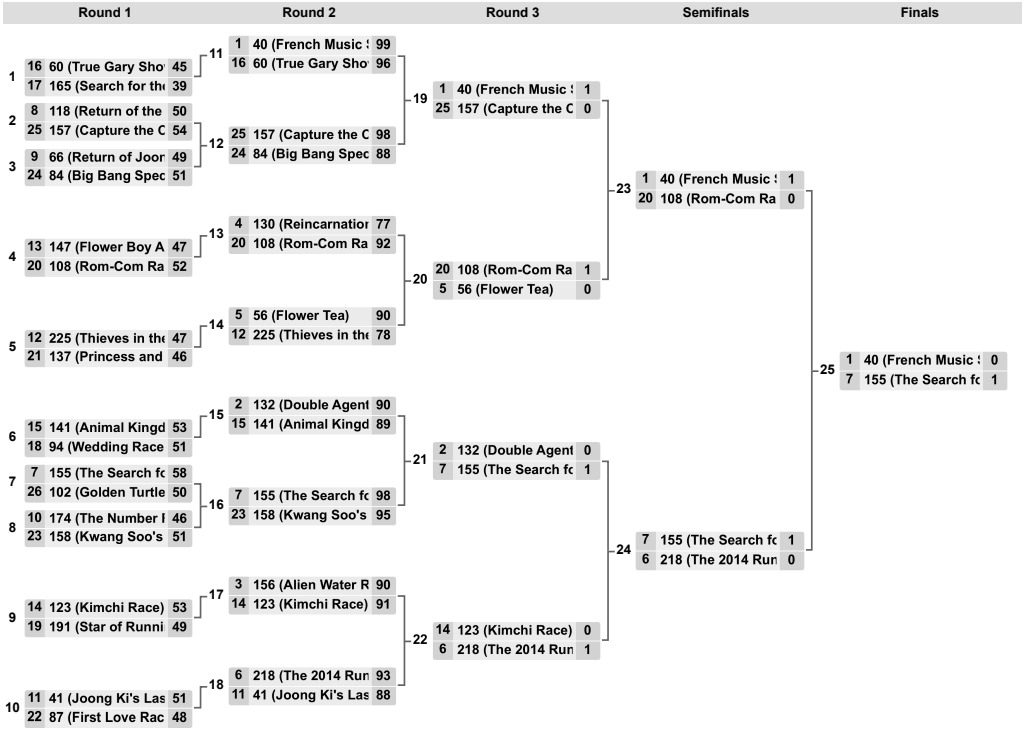
Obr. 1.10: Príklad vyraďovacieho systému

1.5.2.1 Vyraďovací systém na dve porážky

Oproti klasickému vyraďovaciemu systému tu jedna porážka ešte neznamená vyradenie z turnaja. Ako názov vraví na vyradenie z turnaja sú potrebné dve prehry. Výhody tejto varianty sú:

- 3. miesto je možné určiť bez špeciálneho zápasu o toto umiestnenie
- Pokiaľ nie je možné určiť nasadenie tímov, môže sa stať, že dva najsilnejšie tímy by sa stretli hneď v prvom kole a jedno by muselo turnaj opustiť, takto má šancu stále sa dostať do finále
- Každý tím odohrá minimálne 2 hry a $\frac{3}{4}$ tímov aspoň 3 hry, čím sa čiastočne môže odstrániť problém s minimálnym počtom odohratých hier

Nevýhodou môže byť minimálny počet celkových hier, ktorý je pre N účastníkov až $2N - 2$ resp. $2N - 1$ v prípade, že víťaz ani raz neprehral.



Obr. 1.11: Príklad vyraďovacieho systému na dve porážky

1.5.2.2 McIntyrov systém

Jedná sa o súbor systémov, konkrétne 5 systémov určených pre najlepších 4, 5, 6 alebo 8 tímov. Všeobecne sa dá povedať, že sú to vyraďovacie systémy, ktoré dávajú výhodu tímom, ktoré sú do turnaja nasadené vyššie.

Page-McIntyrov systém Je to systém pre 4 najlepšie tímy. Prvé kolo hrajú protisebe dva najvyššie nasadené tímy a víťaz tohto zápasu smeruje do finále. Tím, ktorý prehrá bude hrať s víťazom duelu posledných dvoch tímov a druhú miestenku do finále. Preto sa hovorí aj o „dvojitej šanci“ pre dva najlepšie tímy. Ak predpokladáme, že každý tím ma rovnakú šancu na výhru v každom zápase, pravdepodobnosť, že turnaj vyhrá 1. alebo 2. tím je 37,5 %, zatiaľ čo pravdepodobnosť pre 3. a 4. tím je 12,5 %.

Kolo	Zápas	Názov	Tím 1		Tím 2
1	A	1. semifinále	Rank 3	v	Rank 4
	B	2. semifinále	Rank 1	v	Rank 2
2	C	predfinále	Porazený B	v	Víťaz A
3	D	finále	Víťaz B	v	Víťaz C

Tabuľka 1.2: Page-McIntyrov systém

McIntyrov systém pre najlepších 5 tímov Ako názov hovorí je to systém pre nepárny počet tímov, konkrétne 5. V prvom kole má najvyššie nasadený tím voľno. Posledné dva tímy hrajú o to, kto v turnaji zostáva a kto skončí. Druhý a tretí tím zas hrajú o to, ktorý zápas druhého kola budú hrať. Ďalšie kolá už sú podobné ako v predchádzajúcom systéme pre 4 tímy. Pravdepodobnosti výhier v turnaji sú 37,5 % pre tím najvyššie nasadený, 25 % pre 2. a 3. tím a pre zvyšné dva 6,25 %.

Kolo	Zápas	Názov	Tím 1		Tím 2
1	A	eliminačný zápas	Rank 4	v	Rank 5
	B	kvalifikačný zápas	Rank 1	v	Rank 2
2	C	1. semifinále	Porazený B	v	Víťaz A
	D	2. semifinále	Rank 1	v	Víťaz B
3	E	predfinále	Porazený D	v	Víťaz C
4	F	finále	Víťaz D	v	Víťaz E

Tabuľka 1.3: McIntyrov systém pre najlepších 5 tímov

Prvý McIntyrov systém pre najlepších 6 tímov Tento systém je totožný s predchádzajúcim, až na to, že je samozrejme pre 6 tímov a tým pádom v prvom kole hrajú všetky tímy a vypadnú hneď dva, nie len jeden. Spomínanú dvojitú šancu tu majú hneď 3 tímy.

Kolo	Zápas	Názov	Tím 1		Tím 2
1	A	1. eliminačný zápas	Rank 5	v	Rank 6
	B	2. eliminačný zápas	Rank 3	v	Rank 4
	C	kvalifikačný zápas	Rank 1	v	Rank 2
2	D	1. semifinále	Porazený C	v	Víťaz A
	E	2. semifinále	Víťaz C	v	Víťaz B
3	F	predfinále	Porazený E	v	Víťaz D
4	G	finále	Víťaz E	v	Víťaz F

Tabuľka 1.4: Prvý McIntyrov systém pre najlepších 6 tímov

Druhý McIntyrov systém pre najlepších 6 tímov Druhý špecializovaný systém pre 6 tímov poupravuje prvý tak, že odstraňuje nevýhodu pre 4. tím, ktorý by hral proti ťažšiemu súperovi ako 5. tím. Oba tieto systémy majú však nevýhodu v tom, že porazený z najťažšieho zápasu 1. kola (prvý prot druhému) sa ocitne v druhom kole v zápase, v ktorom porazený z turnaja vypadáva.

Kolo	Zápas	Názov	Tím 1		Tím 2
1	A	1. eliminačný zápas	Rank 4	v	Rank 5
	B	2. eliminačný zápas	Rank 3	v	Rank 6
	C	kvalifikačný zápas	Rank 1	v	Rank 2
2	D	1. semifinále	Porazený C	v	Nižšie postavený víťaz A, B
	E	2. semifinále	Víťaz C	v	Vyššie postavený víťaz A, B
3	F	predfinále	Porazený E	v	Víťaz D
4	G	finále	Víťaz E	v	Víťaz F

Tabuľka 1.5: Druhý McIntyrov systém pre najlepších 6 tímov

McIntyrov systém pre najlepších 8 tímov Obsahuje kombináciu predchádzajúcich systémov, kde dostanú druhú šancu len dvaja najvyššie nasadení porazení z 1. kola. Pravdepodobnosti na výhru sú 18,75 % pre prvého a druhého, 15,625 % pre tretí tím, 12,5 % pre 4. a 5., 9,375 % 6. a 6,25 % pre 7. a 8.

Kolo	Zápas	Názov	Tím 1		Tím 2
1	A	1. kvalifikačný zápas	Rank 4	v	Rank 5
	B	2. kvalifikačný zápas	Rank 3	v	Rank 6
	C	3. kvalifikačný zápas	Rank 2	v	Rank 7
	D	4. kvalifikačný zápas	Rank 1	v	Rank 8
2	E	2. semifinále	4. najvyššie postavený víťaz A, B, C, D	v	2. najvyššie postavený porazený A, B, C, D
	F	1. semifinále	3. najvyššie postavený víťaz A, B, C, D	v	1. najvyššie postavený porazený A, B, C, D
3	G	2. predfinále	2. najvyššie postavený víťaz A, B, C, D	v	Víťaz F
	H	1. predfinále	1. najvyššie postavený víťaz A, B, C, D	v	Víťaz E
4	I	finále	Víťaz G	v	Víťaz H

Tabuľka 1.6: McIntyrov systém pre najlepších 8 tímov

1.5.3 Švajčiarsky systém

Prvýkrát bol tento systém použitý na šachovom turnaji v Zürichu roku 1895, odtiaľ pochádza názov. Je bežne používaný v niektorých športoch.

Nejde o vyraďovací systém. Princíp spočíva v rozdelení hráčov tak, aby v každom kole proti sebe hrali rovnako alebo aspoň podobne kvalitné tímy. Párovanie tímov do zápasov je v prvom kole vyriešené buď losovaním alebo na základe hodnotenia tímov. Tím za výhru dostane 1 bod a v ďalších kolách potom proti sebe hrajú tímy s rovnakým alebo približne rovnakým počtom bodov. V prípade, že má viacero tímov rovnaké skóre, zoradia sa podľa ich nasadenia do turnaja a tímy z prvej polovice potom vždy hrajú proti tímu z druhej polovice. Ak má napríklad 8 tímov rovnaké skóre, v ďalšom kole bude hrať proti sebe 1. tím a 5. tím. Toto pravidlo sa poruší jedine ak tímy už proti sebe predtým hrali.

Na určenie celkového víťaza stačí $\lceil log_2(N) \rceil$ zápasov podobne ako vo vyraďovacom systéme. Ak by sa hralo menej zápasov bolo by možné, že by dva tímy mali rovnaké skóre a ešte spolu nehrali. Výhodou tohto systému však je, že tímy sa nevyraďujú a tak majú všetky rovnaký počet zápasov, čo je v kanoepole žiadúce. Navyše je možné určiť jednoznačné poradie pre všetky tímy, nie len víťaza. Na začiatku sa určí počet kôl a ak majú po poslednom kole nejaké tímy rovnaké skóre, poradie určí pomocné pravidlo, najčastejšie *Buchholzovo*, ktoré uprednostňuje tímy, ktoré hrali proti silnejším súperom. Nevýhodou môže byť, že turnaj nekončí vždy vyvrcholením turnaja, zápasom o 1. miesto, pretože sa môže stať, že je niektorý z tímov taký dominantný, že získa nad ostatnými dostatočnú prevahu, aby ho v skóre už nedobehli.

Môj softvér pre generovanie herného plánu bude mať pre organizátorov určite na výber všetky spomínané systémy, pretože si myslím, že aj keď pravidelné turnaje majú väčšinou stabilný nemeniaci sa systém, občasná zmena môže zvýšiť atraktivitu takejto súťaže a odstrániť stereotyp.

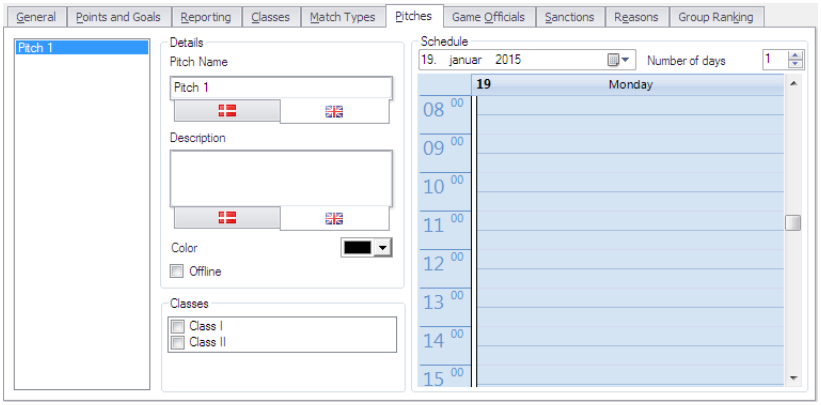
1.6 TournaManage

TournaManage[2] je program aktuálne používaný na rozvrhovanie turnajov v kanoepole. Slúži výborne ako inšpirácia a zároveň je aj zdrojom chýb, ktorým sa chcem vyvarovať. TournaManage pozostáva z dvoch samostatných komponentov. Existuje samostatná server[21] aplikácia a program, ktorý slúži ako klient[22]. Serverová časť má na starosti nastavenia dátového úložiska a obsahuje engine pracujúci s dátami. Klientovou časťou sa potom dá pripojiť na bežiaci server a spravovať turnaje, vytvárať herné systémy a podobne. Táto architektúra má svoju myšlienku - organizátor vytvorí turnaj, vyplní dôležité prvky a k serveru sa pomocou klienta pripoja zástupcovia jednotlivých tímov a môžu sa prihlásiť do turnaja alebo sa prihlási rozhodca, ktorý má obmedzené práva a zadá výsledok zápasu do systému.

Je to pekný nápad ale v skutočnosti je to zbytočne komplikované. Bežný workflow s týmto programom vyzerá tak, že o všetko sa stará jeden, maximálne dvaja ľudia z tímu organizátorov. Prihlášky do turnajov sa riešia inou cestou, napríklad cez e-mail a zodpovedná osoba potom všetko vloží do databázy. Preto je celkom zbytočné mať aplikáciu rozdelené na dve časti. Proces vytvárania turnaja je potom zbytočne komplikovaný a je potrebné striedavo zapínať a vypínať serverovú a klientsku časť aplikácie.

Ďalší nedostatok tohto softvéru je, že funguje len pod operačným systémom Microsoft Windows. Najmä v dnešnej dobe, kedy už nie je Windows jediným operačným systémom drvivej väčšiny ľudí a ďalšie systémy stále získavajú na popularite je veľmi vhodné aby boli programy multiplatformné. To je dôležité hlavne pre programy, ktoré sú vysoko špecializované a nie je mnoho konkurenčných riešení pre každú platformu.

Asi najväčším nedostatkom je ale fakt, že vytváranie aj jednoduchého turnaja pozostáva z príliš veľa záložiek a jednotlivé položky, ktoré sú niekedy nepodstatné sú nastavené ako požadované a ich vypĺňanie potom zbytočne predlžuje proces vytvárania turnaja a celé používateľské rozhranie pôsobí veľmi neprehľadne a komplikovane. Takisto nie je dostupná databáza tímov takže pre každý jeden turnaj je potrebné vytvoriť tímy s jednotlivými hráčmi a nastaviť všetky detaily. Veľmi veľa z týchto položiek sa nemení ako napríklad hracia doba. Je však dobré ponechať možnosť v krajnom prípade aj tieto položky zmeniť. Ako najlepšia voľba je podľa mňa nastaviť takéto hodnoty na obvyklú hodnotu a skryť ju z hlavného okna programu. Celé UI sa vďaka tomu prečistí a vo väč šine prípadov aj urýchli tvorbu herného systému. Všetky takéto položky by potom bolo možné nastaviť v samostatnej záložke.



Obr. 1.12: Ukážka UI programu TournaManage

Kapitola 2

Riešenie

2.1 Formulácia problému

Prvý logický krok je formulovanie obmedzení. Zo zadania turnaja pre kanoepolo som vyvodil 8 obmedzujúcich podmienok, ktoré je nutné splniť. Z toho 7 podmienok musí byť nutne splnených a posledná 8. musí byť splnená v ideálnom prípade. Týka sa totižto viac než dvoch premenných a tím pádom výraznejšie ovplyvňuje možné riešenie.

O1 Na každom ihrisku sa môže v jednu chvíľu hrať práve jedno strenutie

O2 Každého zápasu sa účastnia 2 odlišné tímy z rovnakej kategórie

O3 Každý zápas rozhoduje rozhodca z tímu odlišného od oboch hrajúcich tímov

O4 Zápas kategórie A môže rozhodovať len rozhodca z tímu kategórie A

O5 Každý tím musí hrať aspoň 4 hry ak turnaj trvá 1 deň

O6 Každý tím musí hrať aspoň 3 hry za deň ak turnaj trvá 2 dni

O7 Žiadny tím nemôže hrať 2 zápasy po sebe

O8 Tím, ktorý rozhoduje zápas nemôže hrať alebo rozhodovať zápas tesne pred alebo tesne po tomto zápase

Je jasné, že pre splnenie obmedzenia *O8* už je potrebné väčšie množstvo tímov na turnaji. Preto ak by neexistovalo žiadne riešenie pre podmienku *O8*, bude vypustená z množiny podmienok. Takisto by bolo možné

pochybovať o splniteľnosti obmedzenia $O7$, napríklad pri turnaji s počtom účastníkov 4. V takom prípade ale nemá význam riešiť naplánovanie hracieho plánu pomocou CSP. Za pár minút sa dá navrhnúť plán manuálne. Môj softvér samozrejme bude univerzálny a bude fungovať aj pre krajné prípady. Tých je veľmi obmedzený počet, konkrétne turnaj pre 3 a 4 tímy, preto v tomto prípade bude predpripravený hotový plán. V skutočnosti sú takéto miniatúrne turnaje veľmi zriedkavé ale keď sa už objavia väčšinou sa rieši problém dvoch zápasov jedného tímu po sebe dlhšou prestávkou medzi zápasmi. Ak je počet tímov ≥ 5 už existuje rozloženie zápasov také, že jeden tím nemusí hrať dva zápasy po sebe.

CSP je definovaný ako trojica množín X - množina premenných, D - množina domén, C - množina obmedzení. Aby som mohol definovať konkrétne obmedzujúce podmienky je naprv dôležité stanoviť, čo budú predstavovať premenné. V mojom prípade to budú jednotlivé časové okná prichystané pre zápasy. Každé ihrisko má organizátorom turnaja stanovený čas od-do, kedy je možné hrať zápasy. Je vyhradený čas 30 minút pre jeden zápas vrátane krátkej prestávky. Majme teda ihrisko $I1$, na ktorom sa môžu hrať zápasy napríklad od 9:00 do 12:00. Vznikne nám tak celkovo 6 premenných, kde ku každej je treba priradiť 3 tímy - 2 hrajúce + 1 rozhodcovský, tak aby vyhovovali podmienkam.

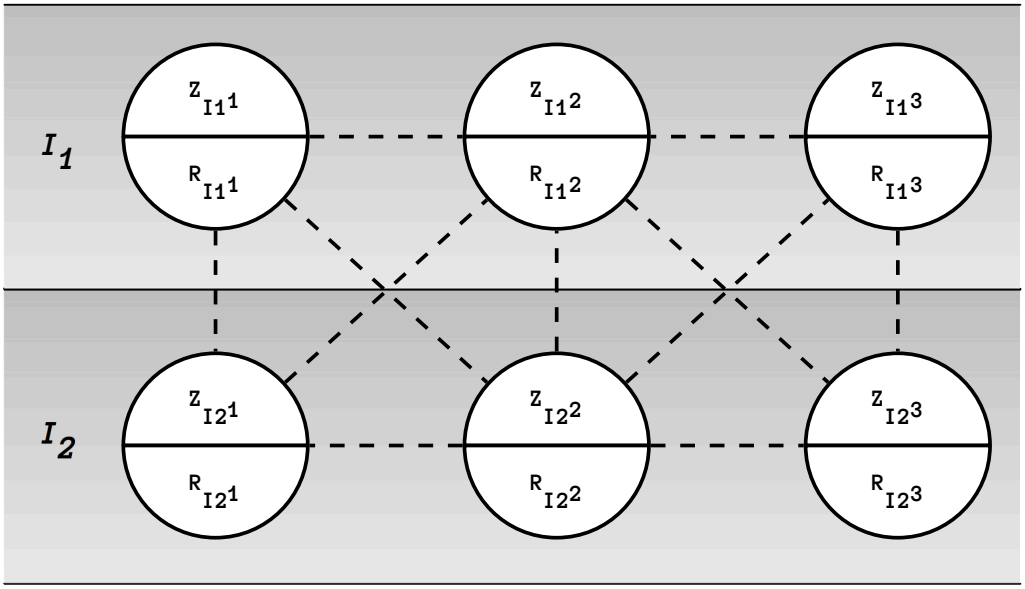
Okrem hrajúcich tímov je však potrebné roztriediť aj rozhodcovské tímy. Preto bude mať každá premenná predstavujúca zápas ešte pomocnú premennú, ktorá bude slúžiť práve pre určenie rozhodcu zápasu. Samozrejme medzi touto dvojicou premenných bude vzťah, ktorý predstavím neskôr.

Ďalej je potrebné definovať domény premenných, teda hodnoty, ktoré bude možné premenným priraďovať. V kapitole o herných systémoch som spomenul všetky dostupné a herné systémy, ktoré budú dostupné. Keďže každý herný systém je presne definovaný má aj presne definované zápasy, ktoré sa budú hrať. Je potrebné len určiť ich poradie tak, aby boli konzistentné. Domény pre všetky premenné budú teda tvorené hodnotami, ktoré predstavujú konkrétne zápasy. Napríklad ak máme kategóriu A so 4 tímami a kategóriu B so 4 tímami domény budú tvorené množinou $M = \{A12, A13, A14, A23, A24, A34, B12, B13, B14, B23, B24, B34\}$. Domény pomocných premenných bude zas množina všetkých tímov $N = \{A1, A2, A3, A4, B1, B2, B3, B4\}$.

Keď už sú definované premenné aj ich domény zostáva k nim dodefinovať obmedzenia podľa zadania. Obmedzenie $O2$ je zabezpečené vďaka tomu, ako som definoval domény. Obmedzenie $O1$ je zas zabezpečené tým, ako som definoval premenné a tým, že každej premennej je možné priradiť len jednu hodnotu - jeden zápas. Podmienky $O5$ a $O6$ sú zabezpečené vďaka hernému systému, ktorý určuje koľko zápasov bude každý tím hrať.

Zvyšné podmienky hovoria už o vzťahu dvoch alebo viacerých premenných. Celý problém možno zobraziť ako graf, kde jeden zápas bude predstavovať dvojica uzlov. Jeden uzol pre premennú zápasu a druhý pre premennú rozhodcu. Keďže obmedzenie $O3$ hovorí, že rozhodcovský tím musí byť odlíšný od oboch hrajúcich tímov, budú oba uzly spojené hranou, ktorá znamená, že hodnota rozhodcovskej premennej sa nesmie rovnať hodnote ani jedného z hrajúcich tímov. Ak je napríklad zápas $A23$ tak rozhodcovská premenná nesmie mať hodnotu $A2$ ani $A3$. Bude to fungovať tak, že keď sa k zápasovej premennej priradí hodnota, z domény rozhodcovskej premennej spojenej s týmto uzlom hranou sa dočasne odstránia hodnoty predstavujúce oba tímy alebo opačne. Pre splnenie obmedzenia $O4$ sa v prípade priradenia zápasu kategórie A odstránia nie len hrajúce tímy ale aj všetky tímy z ostatných kategórií. Navyše vzťahy samozrejme existujú aj medzi ďalšími uzlami. Konkrétne budú hranou spojené uzly zápasov pokiaľ sa odohrávajú v jeden čas a takisto zápasy, ktoré sa odohrávajú po sebe. Pre zápasové uzly bude teda platiť globálne obmedzenie, ktoré sa často v CSP využíva, *AllDifferent*, ktoré vraví, že všetky premenné musia mať odlišnú hodnotu. To platí keďže žiaden zápas sa nehrá dvakrát. Pre rozhodcov to platiť nemusí, pretože jeden tím môže rozhodovať aj viac rôznych zápasov. Preto hrany medzi uzlami rozhodcov a zápasov znamenajú nerovnosť, teda že rozhodcovský tím sa musí líšiť od oboch hrajúcich.

Ďalšie pravidlá sa použijú v pre-processingu. Podľa želania organizátora môže byť napríklad nejaké ihrisko vyhradené len pre zápasy konkrétnej kategórie, napríklad juniorov, z akýchkoľvek dôvodov. V tom prípade sa v pre-processingu odstránia z domén zápasových premenných všetky zápasy okrem zápasov kategórie juniorov. V prípade, že by sa jednalo o vyhradenie pre kategóriu A, hneď sa odstránia aj hodnoty z domén pre rozhodcovské premenné. Môže byť napríklad prianie otvoriť turnaj konkrétnym atraktívnym zápasom na konkrétnom ihrisku a to zas umožní hneď priradiť fixne jednu hodnotu a prečistiť ostatné domény.



Obr. 2.1: Príklad grafu pre 2 ihriská s 3 časovými oknami pre zápasy - I_i je $i-té$ ihrisko, $Z_{I_i}j$ je $j-ty$ zápas na ihrisku I_i a $R_{I_i}j$ je premenná pre rozhodcu, ktorý rozhoduje $j-ty$ zápas na ihrisku I_i. Uzly v jednom rade teda predstavujú zápasy s rozhodcami na jednom ihrisku. Hranou sú navzájom spojené všetky uzly v jednom stĺpci a v stĺpcoch susedných. Tieto hrany predstavujú obmedzenie, že jeden tím, či už hrajúci alebo rozhodujúci, sa nemôže zúčastniť dvoch zápasov po sebe. V prípade, že sa nejakej premennej priradí hodnota, táto hodnota sa odstráni z domén všetkých premenných, ktoré sú s ňou spojené hranou.

2.2 Databázový model

Návrh databázového modelu obsahuje celkom 12 entít predstavujúcich dôležité objekty pre realizáciu softvéru na generovanie herných systémov pre turnaje.

2.2.1 Entity

TURNAJ

Entita zastupujúca turnaje s najdôležitejšími atribútami. Má $1:N$ reláciu s entitou *IHRISKO*, keďže na turnaji je bežne viac hracích plôch. Ďalšia relácia $1:N$ s entitou *UCAST* predstavuje účasť jednotlivých tímov na

tomto turnaji.

IHRISKO

Táto entita predstavujúca každé ihrisko na turnaji má dôležité atribúty - *Farba* je označenie ihriska farbou pre lepšiu orientáciu. *Popis* a *GPS* nie sú nutne povinné, ale môžu takisto zlepšiť orientáciu tímov najmä v prípade, že sú ihriská od seba viac vzdialené alebo ak ich je väčší počet. Okrem relácie s entitou *TURNAJ* obsahuje $1 : N$ reláciu s entitou *HRACI_DEN* keďže každý deň turnaja môže mať ihrisko rôznu časovú dostupnosť. To sa využíva hlavne v posledný deň turnaja, keď je žiadúce, aby turnaj skončil skôr alebo aby sa simultánne s finálovým zápasom nehral žiadny iný.

HRACI_DEN

Predstavuje dôležitú súčasť turnajov, dovoľuje organizátorom nastaviť jednotlivým ihriskám pre každý deň rôznu časovú dostupnosť. Má $1 : N$ reláciu s entitou *ZAPAS*, keďže samozrejme každý deň na každom ihrisku sa väčšinou odohrá viac zápasov.

UCAST

UCAST je slabá entita, ktorú identifikujú kľúče z entít *TURNAJ* a *TIM*, pričom obsahuje dôležitý atribút *seed*. Seed znamená nasadnie tímu do turnaja, kvalitnejšie tímy majú nižšie číslo seedu. Dôležitý údaj pre rozvrhovanie tímov do herného systému, aby bol čo najférovejší. Ďalej má dve veľmi dôležité relácie s entitami *ROZHODUJE* a *JE_V_ZOSTAVE*. Pomocou *JE_V_ZOSTAVE* je možné identifikovať hráčov, ktorí v tíme hrali na konkrétnom turnaji. Ak by mal *HRAC* priamu reláciu s *TIM*, znamenalo by to, že akákoľvek zmena v zostave by sa prejavila aj spätne, čo nie je veľmi šikovné. Takto sú možné akékoľvek presuny hráčov medzi tímami bez toho, aby vznikali nežiadúce efekty. Podobne relácia s *ROZHODUJE* je potrebná pre evidovanie kategórií, ktorých zápasy môže tím rozhodovať. Toto sa môže líšiť medzi turnajmi.

TIM

Entita *TIM* predstavuje každý tím, nezávisle na kategórii, v ktorej pôsobí. Túto kategóriu predstavuje relácia $1 : 1$ s entitou *KATEGORIA*. *TIM* má atribúty *Nazov* - názov tímu, *Skratka* - skratka tímu, ktorú je vhodné používať v prehľadných tabuľkách a rozpisoch turnajov, *Farba1* a *Farba2* -

primárna a sekundárna farba tímu, pomáha tímom pripraviť sa na každý zápas, aby sa nestalo, že majú tímy ťažko rozpoznateľné dresy. Má až trojitú $1 : 1$ reláciu s entitou *ZAPAS*. Dve predstavujú dva tímy, ktoré spolu súperia a tretia je pre tím, ktorý zápas rozhoduje. Ďalej má reláciu s entitou *UCAST*. Táto relácia znamená účasť daného tímu na turnaji a takisto je vďaka nej možné zistiť, ktoré kategórie zápasov môže tím rozhodovať a ktorí hráči v tíme hrali, na rôznych turnajoch.

KATEGORIA

KATEGORIA je entita predstavujúca kategórie, v ktorých tímy na turnaji súťažia. Tie sa môžu líšiť turnaj od turnaja, takisto ako pravidlo pre rozhodovanie medzi jednotlivými kategóriami.

ZAPAS

Entita *ZAPAS*, ktorá samozrejme predstavuje zápas turnaja má okrem spomínaných relácii ďalšie dve $1 : N$ s entitami *KARTA* a *GOL* dôležité pre identifikovanie hráčov, ktorí v zápase skórovali alebo dostali kartu. Relácie s *TIM* a *HRACI_DEN* jednoznačne určujú zápas - na ktorom turnaji, ktorý deň na ktorom ihrisku a ktorí hráči ktorých tímov tento zápas odohrali.

HRAC

HRAC má atribúty dôležité pre identifikáciu jednotlivých hráčov a dve $1 : N$ relácie pre identifikovanie koľko gólov skórovali a koľko kariet dostali v jednotlivých zápasoch. Relácia s *JE_V_ZOSTAVE* definuje za ktorý tím hral na ktorom turnaji.

KARTA

KARTA predstavuje varovanie vydané rozhodcom v zápase pre konkrétneho hráča. Atribút typ je dôležitý na určenie konsekvencií, ktoré z tejto karty vyplývajú pre hráča alebo pre tím.

GOL

GOL je samozrejme najdôležitejší na určenie víťaza zápasu poprípade najlepšieho strelca turnaja. Obe posledné entity potrebujú okrem identifikátora zápasu a hráča aj vlastný unikátny identifikátor keďže v jednom zápase je možné, aby jeden hráč dal viac gólov alebo dostal viac kariet. Nepomohlo

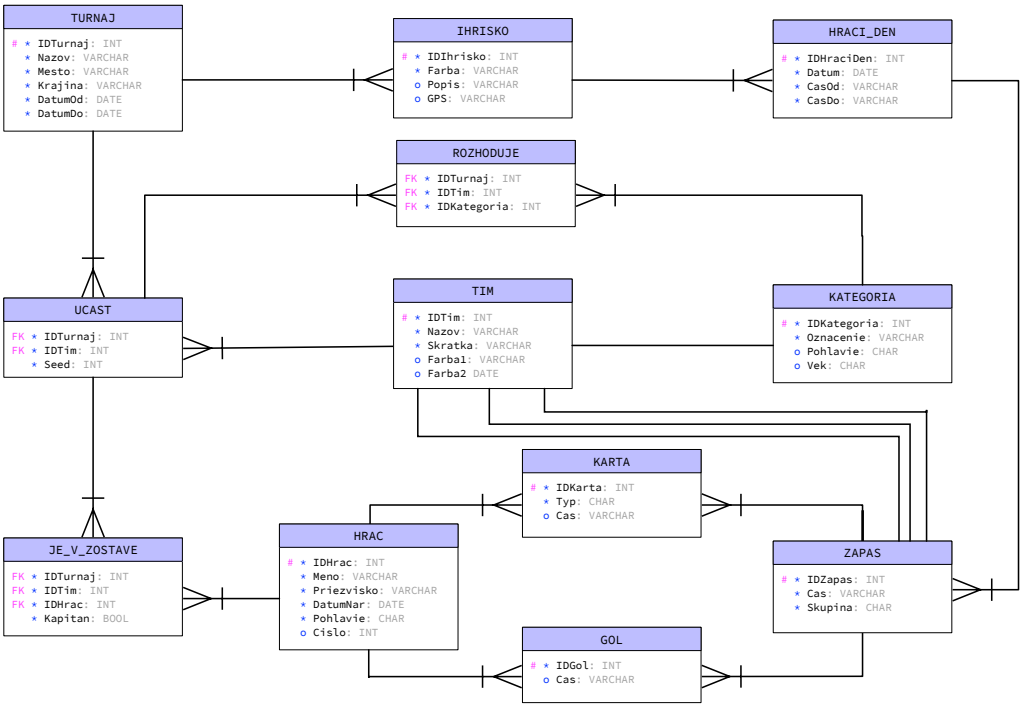
by ani urobiť z atribútu minuta kľúč pretože nie je vylúčené, že sa tieto udalosti stanú v rovnakej minúte.

ROZHODUJE

Slabá entita dôležitá, aby bolo možné rozoznať aké kategórie smie tím rozhodovať na rôznych turnajoch.

JE_V_ZOSTAVE

Spája hráčov a tímy. Hráči môžu časom samozrejme prestupovať medzi klubmi, preto nie je vhodná priama relácia s tímom. Atribút *Kapitan* určuje, či je hráč v tíme na konkrétnom turnaji kapitánom.



Obr. 2.2: Návrh logického databázového modelu

Záver

Cieľom tejto práce bolo vykonať rozsiahly rešerš v oblasti plánovania športových turnajov. Ten bude slúžiť ako podklad pre implementáciu nového softvéru na vytváranie turnajových herných plánov. Zoznámil som sa, a zanalyzoval som vhodné algoritmy pre riešenie takéhoto problému. Z kandidátov - problém s obmedzujúcimi podmienkami, pseudo-booleovská optimalizácia a lineárne programovanie vyšla ako najvhodnejšia metóda pre implementáciu prvá menovaná. Vybral som ju na základe preštudovaných prác, ktoré sa danou problematikou zaoberajú. Pomocou nej som problém formálne definoval - stanovil som dôležité obmedzujúce podmienky, a naznačil spôsob riešenia konkrétnej úlohy. Možnosť využitia on-line algoritmu som pre tento typ problému zamietol. V porovnaní s off-line ekvivalentmi má nižšiu efektivitu. Zoznámil som sa s momentálne jediným softvérovým riešením pre vytváranie turnajov v kanoepole. Vyvodil som z toho závery, ktoré pomôžu vyvarovať sa chybám pri samotnej implementácii. Navrhol som databázový model, ktorý bude fungovať lepšie ako existujúci model v programe TournaManage.

V práci by som sám rád pokračoval a vytvoril plne funkčný softvér. Táto práca mi umožní plne sa venovať kvalitnému softvérovému návrhu a implementácii. Vhodné by bolo taktiež vytvoriť aj webovú verziu, poprípade mobilnú aplikáciu.

Literatúra

[1] Canoe Polo Competition Rules [online]. 2015, [cit. 9. 5. 2017]. Dostupné z: http://www.canoagem.org.br/arquivos/ckfinder/files/canoe%20polo%20rules%202015.pdf

[2] TournaManage v. 2. Manual [online]. [cit. 9. 5. 2017]. Dostupné z: http://www.tournamanage.net/docs/doku.php?id=tm-manual

[3] Anson, S.; Lester, S.: Sports Scheduling: Algorithms and Applications [online]. [cit. 9. 5. 2017]. Dostupné z: https://courses.cs.washington.edu/courses/csep521/07wi/prj/sam_scott.pdf

[4] Henz, M.; Müller, T.; Thiel, S.: Global constraints for round robin tournament scheduling. *European Journal of Operational Research* [online], ročník 153, č. 1, 2004: s. 92 – 101, ISSN 0377-2217, doi:https://doi.org/10.1016/S0377-2217(03)00101-2, [cit. 9. 5. 2017]. Dostupné z: http://www.sciencedirect.com/science/article/pii/S0377221703001012

[5] Nurmi, K.; Kyngäs, J.; Goossens, D.: Scheduling a triple round robin tournament for the finnish national ice hockey league for players under 20. In *2011 IEEE Symposium on Computational Intelligence in Scheduling (SCIS)* [online], April 2011, s. 46–53, doi:10.1109/SCIS.2011.5976545, [cit. 9. 5. 2017]. Dostupné z: http://ieeexplore.ieee.org/document/5976545

[6] Leong, G. T.: Constraint Programming for the Travelling Tournament Problem [online]. Technická zpráva, National University of Singapore, 2003, [cit. 9. 5. 2017]. Dostupné z: http://www.comp.nus.edu.sg/~henz/students/gan_tiaw_leong.pdf

[7] Carlsson, M.; Johansson, M.; Larson, J.: Scheduling double round-robin tournaments with divisional play using constraint programming. *European Journal of Operational Research* [online], ročník 259, č. 3, 2017: s. 1180 – 1190, ISSN 0377-2217, doi:https://doi.org/10.1016/j.ejor.2016.11.033, [cit. 9. 5. 2017]. Dostupné z: http://www.sciencedirect.com/science/article/pii/S0377221716309584

[8] Easton, K.; Nemhauser, G.; Trick, M.: *The Traveling Tournament Problem Description and Benchmarks*, kapitola 1. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, ISBN 978-3-540-45578-3, s. 580 – 584, doi:10.1007/3-540-45578-7_43, [cit. 9. 5. 2017]. Dostupné z: http://dx.doi.org/10.1007/3-540-45578-7_43

[9] Thielen, C.; Westphal, S.: Complexity of the traveling tournament problem. *Theoretical Computer Science* [online], ročník 412, č. 4, 2011: s. 345 – 351, ISSN 0304-3975, doi:http://dx.doi.org/10.1016/j.tcs.2010.10.001, [cit. 9. 5. 2017]. Dostupné z: http://www.sciencedirect.com/science/article/pii/S0304397510005451

[10] Bhattacharyya, R.: Complexity of the Unconstrained Traveling Tournament Problem. *Operations Research Letters* [online], ročník 44, č. 5, 2016: s. 649 – 654, ISSN 0167-6377, doi:https://doi.org/10.1016/j.orl.2016.07.011, [cit. 9. 5. 2017]. Dostupné z: http://www.sciencedirect.com/science/article/pii/S0167637716300682

[11] CSP Tutorial [online]. 2005, [cit. 9. 5. 2017]. Dostupné z: http://4c.ucc.ie/web/outreach/tutorial.html

[12] Barták, R.: Constraint Satisfaction [online]. 1998, [cit. 9. 5. 2017]. Dostupné z: http://ktiml.mff.cuni.cz/~bartak/constraints/constrsat.html

[13] Farhang, Y.; Meybodi, M. R.; Hatamlou, A. R.: Improving the Efficiency of Forward Checking Algorithm for Solving Constraint Satisfaction Problems. In *2008 Eighth International Conference on Intelligent Systems Design and Applications* [online], ročník 1, Nov 2008, ISSN 2164-7143, s. 240–245, doi:10.1109/ISDA.2008.272, [cit. 9. 5. 2017]. Dostupné z: http://ieeexplore.ieee.org/document/4696210/

[14] Russell, S.; Norvig, P.: Constraint Satisfaction Problems. In *Artificial Intelligence: A Modern Approach* [online], Prentice Hall, druhé vydání, 2002, s. 137 – 160, [cit. 9. 5. 2017]. Dostupné z: http://aima.cs.berkeley.edu/2nd-ed/newchap05.pdf

[15] Boros, E.; Hammer, P. L.: Pseudo-Boolean optimization. *Discrete Applied Mathematics* [online], ročník 123, č. 1–3, 2002: s. 155 – 225, ISSN 0166-218X, doi:https://doi.org/10.1016/S0166-218X(01)00341-9, [cit. 9. 5. 2017]. Dostupné z: `http://www.sciencedirect.com/science/article/pii/S0166218X01003419`

[16] Ferguson, T. S.: Linear Programming [online]. [cit. 9. 5. 2017]. Dostupné z: `https://www.math.ucla.edu/~tom/LP.pdf`

[17] Durán, G.; Guajardo, M.; Sauré, D.: Scheduling the South American Qualifiers to the 2018 {FIFA} World Cup by integer programming. *European Journal of Operational Research* [online], 2017: s. – , ISSN 0377-2217, doi:https://doi.org/10.1016/j.ejor.2017.04.043, [cit. 9. 5. 2017]. Dostupné z: `http://www.sciencedirect.com/science/article/pii/S0377221717303909`

[18] Karp, R. M.: On-Line Algorithm Versus Off-Line Algorithms: How Much is it Worth to Know the Future? [online]. Technická zpráva, Internation Computer Science Institute, Júl 1992, [cit. 9. 5. 2017]. Dostupné z: `http://www.icsi.berkeley.edu/pubs/techreports/TR-92-044.pdf`

[19] Sleator, D. D.; Tarjan, R. E.: Amortized Efficiency of List Update and Paging Rules. *Communications of the ACM*, ročník 28, č. 2, 1985: s. 202 – 208.

[20] Lucas, É.: Les jeux de demoiselles. *Récréations Mathématiques* [online], 1883: s. 161 – 197, [cit. 9. 5. 2017]. Dostupné z: `https://archive.org/stream/p2rcrationsm00lucauoft#page/176/mode/2up`

[21] TournaManage Server v2.0.80 [software]. [cit. 9. 5. 2017, požiadavky na systém: MS Windows 7 alebo novší, 1 GB RAM, 300 MB]. Dostupné z: `http://tournamanage.net/currentrelease/Server-2.0.80.exe`

[22] TournaManage Client v2.0.170 [software]. [cit. 9. 5. 2017, požiadavky na systém: MS Windows 7 alebo novší, 1 GB RAM, 300 MB]. Dostupné z: `http://tournamanage.net/currentrelease/Client-2.0.170.exe`

Dodatok A

Slovní zadání pro generátor herního systému

Příklad modelové situace ve sportu kanoepolo:

- 4 hřiště
- Prostor na jeden zápas je 30 minut
- 40 týmů, týmy jsou zároveň i rozhodčími
- Každý zápas má přirazen jen jeden tým, který ho rozhoduje
- 5 Kategorii: muži A, muži B, ženy A, junioři U16, junioři U14
- Ženy, děti, kategorie B a nižší nemohou pískat kategorii A, děti mohou pískat samy sebe (pískají jejich trenéři)
- Každý tým musí odehrát 5-8 zápasů za víkend
- Ideální velikost skupiny je 4 týmy (není nutné)
- Nutné vzít v úvahu nesoudělný počet týmů v kategorii (lichý počet týmů)
- Důležité je, aby týmy neměly zápasy hned za sebou a ideálně aby nepískaly zápas hned po/před co odehrají zápas (pauza stačí 30 min)

A.1 Definice a slovní popis dílčích pojmů

A.1.1 Turnaj

Jedno nebo vícedenní sportovní akce, odehraná podle předem dohodnutých pravidel. Tato pravidla mohou ovlivnit výběr vítěze z jednotlivých fází turnaje nebo minimální přestávky mezi dílčími činnostmi na turnaji. Turnaj se odehrává na hřištích.

A.1.2 Hřiště

Turnaj je organizován na předem známém počtu hřišť, kde je dopředu známo časové kvantum dostupné pro každý den turnaje ("otevírací doba", může být každý den jinak - obvykle se poslední den hraje kratší čas než ostatní dny).

A.1.3 Zápas

Základní časová a hodnotící entita na turnaji. Zápasu se účastní:

- dva hrající týmy,
- tým rozhodčích.

Zápas existuje ve variantách:

- vítěz musí být stanoven (např. finálová část turnaje),
- vítěz nemusí být stanoven (např. základní skupina).

Zápas se skládá z volitelného počtu těchto částí:

- hrací část,
- pauza,
- penalty.

Každá z těchto částí může mít různou dobu trvání, mohou být řazeny libovolně za sebou a mohou se opakovat (příklad z kanoepola: hra 10 min, pauza 3 min, hra 10 min, v případě nerozhodného zápasu hra 7 min, pauza 3 min, hra 7 min, pauza 3 min, penalty, penalty).

Každý herní blok se začíná v čase 00:00, pořadí herního bloku se označuje doplňující číslovkou (typicky první a druhý poločas).

Zápas je ohodnocen body pro účastnící se týmy podle situace (s příklady):

- výhra (obvykle 3 body vítěz),
- prohra (obvykle 0 bodů poražený),
- remíza bez prodloužení (1 bod oba týmy),
- výhra v prodloužení (3 body vítěz),
- prohra v prodloužení (0 bod poražený),
- nedostavení se na hřiště (výhra 7:0 a 3 body pro tým, který se dostavil).

Ze zápasu je pořízen protokol, který může obsahovat tyto údaje:

- identifikátory obou hrajících týmů,
- identifikátor týmu rozhodčích,
- čas začátku a konce zápasu,
- informace o vstřeleném gólu (číslo skórujícího hráče, identifikátor teamu, časovou značku od začátku herního bloku),
- informace o přerušeních hry při faulu (barva karty označující faul, číslo faulujícího hráče a identifikátor jeho týmu, časová značka od začátku herního bloku),
- celkové skóre zápasu a přidělení bodů jednotlivým týmům podle výsledků (možno automaticky dopočítat).

A.1.4 Tým

Tým se skládá z 8 hráčů a má přiřazenou výkonostní kategorii. Minimální počet hráčů na tým závisí na organizátorech (4 hráči - bazénový turnaj, 5 hráčů - venkovní turnaj).

A.1.5 Hráč

Hráče definuje jeho jméno (a příjmení), pohlaví, číslo jeho vesty a příslušnost k týmu. Volitelnou částí informací o hráči je jeho datum narození, které může sloužit k automatické kontrole příslušnosti k výkonostní kategorii. Podle pohlaví a data narození lze automaticky odvodit (výkonostní) kategorie pro juniorské týmy, například: U19 pro muže do 19 let, U19W pro ženy do 19 let, U21 a U21W respektivě.

A.1.6 Kategorie

V rámci turnaje spolu soupeří pouze týmy v rámci jedné výkonostní kategorie. Proto je nutné každému týmu přiřadit kategorii.

A.1.7 Herní systém

Základním předpokladem turnaje je vytvoření herního systému tak, aby si každý tým zahrál minimálně 4 hry za turnaj v případě jednodenního turnaje (maximum není omezeno, ale je doporučeno ho držet menší 8). V případě případě vícedenního turnaje je minimální počet her 3 hry denně, poslední den turnaje se tento počet snižuje na 2 hry denně (aby byla časová rezerva na úklid a odjezd účastnících se týmů - je to běžně řešeno "otevírací dobou hřiště").

Generátor může využít sadu předpřipravených herních systémů pro předem určený počet týmů. Generátor může tyto předpřipravené herních systémů dále optimalizovat, případně si může generovat vlastní systémy.

Vygenerovaný herní systém musí být co nejvíce spravedlivý vůči soupeřícím týmům. Možná manuální korekce se provádí pomocí seedingu.

Doporučeným základním herním systémem je „skupina“, která má za úkol „roztřídit“ týmy podle aktuální výkonnosti. V rámci herního systému „skupina“ je nepřípustné, aby se ve skupině nepotkaly pouze „dobré“ nebo pouze „špatné“ týmy. V herním systému „skupina“ se hraje systémem „každý s každým“. Proto je doporučená velikost skupiny 3-6 týmů, ideálně však 4 (protože první den se hraje skupina a bude splněn požadavek na alespoň 3 zápasy denně).

A.1.8 Seeding

Seeding je způsob ovlivnění herního systému „skupina“, kde každému týmu je přiřazeno číslo určující jeho aktuální „výkonnost“ (jak je který tým dobrý). V herní systému „skupina“ by se neměli potkat týmy, jejichž seeding je bližší než 5 (případně jiná specifikovaná hodnota).

A.1.9 Strategie výběru vítěze

Herním systému „skupina“ se obvykle hraje pouze na body získané v zápasích bez prodloužení. Z toho vyplývá možná shoda bodů pro 2 nebo více týmů a je nutné rozhodnout, který tým bude ohodnocen jako lepší (horší). Doporučené strategie:

- získané body,

- počet vyhraných zápasů,
- rozdíl vstřelených/inkasovaných gólů,
- výsledek vzájemného zápasu - vítězství ve skupině mezi týmy,
- výsledek vzájemného zápasu - kolik hráčů dokázalo vstřelit gól,
- výsledek vzájemného zápasu - hráč s největším počtem nastřílených gólů,
- nejvíce vstřelených gólů,
- nejméně inkasovaných gólů,
- náhodný los.

Klíčem k postupu ze skupiny se může uplatnit jedna nebo více strategií. Každé strategii lze přiřadit prioritu, podle které se určí „vítěz“. V případě nerozhodnutelnosti podle strategie s vyšší prioritou se rozhoduje podle strategie s nižší prioritou až do konečného rozhodnutí. Strategie „náhodný“ los je rozhodnutelná vždy.

A.1.10 Soudcování a fauly

V případě kanoepola se vyskytují 3 typy varování/trestů, které jsou signalizovány hráči/hráčům/celému týmu prostřednictvím karet s různými barvami a významy:

- Zelená karta - Varování (za 3 zelené karty je udělena jedna žlutá karta),
- Žlutá karta - Výstraha (za 2 žluté karty následuje jedna červená karta),
- Červená karta - Vyloučení

Udělení každé karty se provádí zápisem do protokolu. „Varování“ (zelená karta) a „Výstraha“ (žlutá karta) jsou platné pouze pro zápas ve kterém jsou uděleny a nepřenášejí se do dalších zápasů. „Vyloučení“ je platné pro celý zápas nebo pro celý zápas a všechny následující zápasy (tj. do konce turnaje).

Vyloučení hráče/hráčů může způsobit neschopnost týmu nastoupit k zápasu, pokud počet hráčů klesne pod minimální počet hráčů (typicky 3).

Dodatok **B**

Zoznam použitých skratiek

BO3 Best-of-3 (Najlepší z 3 zápasov)

BT Backtracking

CSP Constraint satisfaction problem (Problém s obmedzujúcimi podmienkami)

DFS Depth-first search (Vyhľadávanie do hĺbky)

DRRT Double round robin tournament (Turnaj každý s každým na dve kolá)

FC Forward checking

GT Generate and test (Generuj a testuj)

ILP Integer linear programming (Celočíselné lineárne programovanie)

MRV Minimum remaining values (Najmenej zostávajúcich hodnôt)

PBO Pseudo-boolean optimization (Pseudo-booleovská optimalizácia)

RRT Round robin tournament (Turnaj každý s každým)

SRRT Single round robin tournament (Turnaj každý s každým na jedno kolo)

TTP Traveling tournament problem

TSP Traveling salesman problem

UI User interface (užívateľské rozhranie)

Dodatok C

Obsah priloženého CD

- `src`
  - `thesis` ..... zdrojová forma práce vo formáte LaTeX
    - `res` ..... použité obrázky vo formáte EPS
- `text` ..... text práce
  - `BP_Klement_Michal_2017.pdf` ..... text práce vo formáte PDF