Dokumentace zápočtového programu z Programování I (NPRG030)

Kompromisátor

David Pěgřímek

http://davpe.net

Obsah

1 Úvodem

Motivační příklad

Představme si situaci, kdy jsme si na oběd pozvali tři velmi dobré přátele. A dobří přátelé vyžadují dobrý oběd, dokonce o čtyřech chodech. Přirozeně vyvstává otázka, v jakém pořadí budeme chody servírovat? Zeptáme se tedy našich kamarádů a každý nám odpoví pořadím, v jakém by si přál chody konzumovat. Existuje však ideální pořadí chodů, kterým uspokojíme všechny jedlíky? A pokud ne, v jakém pořadí máme chody servírovat, abychom kamarády co nejvíce potěšily?

Trošku formálněji

Označme si počet přátel N a říkejme jim *rozhodovači* (v angličtině decision makers). Počet chodů si označíme K (obecně to může být počet jakýchkoliv projektů). Pro jednoduchost si chody (projekty) označíme čísly od jedné do K. Každý z rozhodovačů nám udá pořadí K projektů, které seřadí podle svých preferencí. Každý projekt může přitom použít jen jednou (neboli když už jeden chod sní, tak ho nemůže dostat podruhé). Zapišme tato rozhodnutí do matice.

$$\begin{pmatrix} 2 & 1 & 4 & 3 \\ 3 & 2 & 1 & 4 \\ 2 & 1 & 4 & 3 \end{pmatrix}$$

Takové matici budeme říkat *rozhodovací matice* (v angličtině decision matrix).
Našim úkolem je najít *ideální pořadí*, tedy takové pořadí projektů, které by splňovalo požadavky všech rozhodovačů. Z Arrowovy věty [1] však plyne, že takové pořadí neexistuje. Proto se musíme uchylovat k pořadí, které splňuje jen některé požadavky rozhodovačů, čili *kompromisu*.

O programu Kompromisátor

Program Kompromisátor, jak už název naznačuje je určen k hledání takovýchto kompromisů. K tomuto účelu používá čtyři různé metody, kde každá může dávat jiné výsledky. Program byl vytvořen za účelem demonstrace metod hledání kompromisů a to i pro použití běžným uživatelem. Nutností tedy bylo grafické rozhraní, naopak rychlost už nebyla prioritou, neboť program má sloužit spíše k demonstračním účelům, než výpočetním. Z těchto důvodů byl jako jazyk zvolen C# a prostředí .NET 2.0. Nedílnou součástí programu je tato dokumentace, která je rozdělena na část pro uživatele (Ovládání programu) a na část pro programátory (Program zevnitř). Do programu lze jednoduše přidávat nové metody pro hledání kompromisů.

[1] Kenneth Arrow (*1921) Americký ekonom, nositel Nobelovy ceny za ekonomii

2 Ovládání programu

Pro uživatele systému Windows je ve složce *bin* k dispozici přeložený program **Kompromisátor.exe** a ve složce *source* jsou zdrojové kódy a projekt pro Visual Studio. Pokud byste si chtěli program sami přeložit, je k dispozici soubor *makefile*, který však vyžaduje mít nainstalované Mono. Program je kompatibilní s C# 2.0.

2.1 Vstup programu

Vstup programu tvoří čísla $N \in [1, 8]$ a $K \in [1, 8]$ a rozhodovací matice $N \times K$. Program může přijímat vstup třemi způsoby.

1. Ze souboru
   V hlavním menu *Zadání/Načíst ze souboru* se vybere vstupní soubor. Soubor musí být textový a mít příponu *txt*. Na prvním řádku obsahuje čísla N a K, oddělená mezerou. Poté následuje N řádků, každý z obsahuje K čísel od 1 do K, každé právě jednou. V případě špatného formátu souboru, program zahlásí *Chybný formát souboru*. V případě, že N resp. K není v rozsahu 1 až 8 program zahlásí *Proměnná N resp. K je mimo povolený rozsah*.

2. Zadáním v programu
   V hlavním menu se zvolí *Zadání/Nové* a v dialogu se vyberou čísla N a K. Poté se zpřístupní políčka, kde stačí vyplnit hodnoty rozhodovací matice.

3. Náhodně generovaný
   V případě, že už jsou známy rozměry rozhodovací matice, je možné nechat si její hodnoty nechat vygenerovat náhodně pomocí volby *Zadání/Vygenerovat náhodně*.

Program kontroluje, zda je rozhodovací matice korektní (každý řádek je bijekce do množiny $\{1..K\}$). Pokud tato podmínka není splněna, program zahlásí *Rozhodovací matice má chybné řádky* a vypíše čísla řádků matice, která jsou chybná.

2.2 Spouštění metod

Po načtení korektního vstupu, se vám zpřístupní položka menu *Spustit*. Obsahuje seznam čtyř metod, pro rozhodování o kompromisech. Mohou dávat různé výsledky a trvat různě dlouho. Můžete si vybrat, zda spustíte jen jednu z nich nebo všechny.

Stručná charakteristika metod:

Borda-Kendall (BAK) Nejrychlejší a nejjednodušší metoda, avšak často kritizována.

Maximum agreement heuristic model (MAH) Jednoduchá metoda, pomalejší než BAK a paměťově náročnější.

Consensus ranking model (CRM) Pomalá a paměťově náročná metoda.

Hybrid distance-based model (DCM) Podobně jako CRM, pomalá a paměťově náročná metoda.

Metody jsou podrobněji popsány v sekci Algoritmy.

2.3 Výstup

1. V souboru
   Výstupem je seznam $L \in [1, 4]$ řádků (L je počet spuštěných metod), ve formátu třípísmenná zkratka metody, mezera a kompromis metody (K čísel oddělených mezerou). Po zvolení *Zadání/Uložit* se do souboru uloží vstup i výstup, oddělený prázdným řádkem. Vstup se do souboru ukládá proto, že vstupní matice může být zadána jen v programu a uživatel by tak mohl snadno zapomenout, jakou matici do programu zadával. Soubor je možno uložit jak v plaintextové podobě, tak i ve formátu LaTeX.

2. V programu
   Výstup v programu obsahuje navíc vizualizaci a hodnocení ohodnocovací funkci. Vizualizace je aktivována při spuštění metod zvlášť (nezobrazí se, pokud zvolíte *Spustit/Všechny metody*). V rozhodovací matici se světle zelenou barvou označí ty položky, které mají shodné pořadí s kompromisem, který daná metoda našla. Rovněž je podbarvením zvýrazněn název metody, pro kterou se vizualizuje.

   Hodnocení metody si lze představit, jako počet bodů, které daná metoda dostane za kompromis, který vygenerovala. V případě, že se v kompromisu vyskytuje projekt na stejné pozici jako v rozhodovací matici pro libovolného rozhodovače dostane metoda bod. Ty se pak sečtou a výsledný součet se zobrazí napravo od výsledného kompromisu. Jinými slovy ohodnocovací funkce vrací počet zeleně podbarvených položek v rozhodovací matici.

3 Program zevnitř

V této jsou popsány důležité součásti programu a rozebrány použité algoritmy. Nejsou zde popsány zjevné části programu a GUI (které je implementováno pomocí Windows Forms).

Úložiště dat

Třída DataStorage uchovává vstupní a výstupní data. Rovněž umí pracovat se soubory (načítat vstup ze souboru a ukládat vstup a výstup do souboru).

Algoritmy

Všechny algoritmy jsou odvozené z abstraktní třídy *ConsensusAlgorithms*. Tak je zaručeno, že se navenek chovají stejně, lépe se s nimi pracuje a snadno se přidávají nové algoritmy. Než se pustíme do popisu jednotlivých metod, zavedeme si pár definic.

Matice vzdáleností mezi maticemi A a B je matice V, taková že platí $V_{ij} = |A_{ij} - B_{ij}|$.
Frekvenční matice A je matice rozměrů $K \times K$ pro kterou platí, že A_{ij} je počet výskytu projektu i před projektem j v rozhodovací matici.
Součet matice je součet všech prvků matice.

Borda-Kendall (BAK)

Pro každý projekt spočítáme průměrnou hodnotu pořadí, které mu rozhodovači přidělili (neboli v rozhodovací matici pro každý projekt sečteme všechna j kde prvek a_{ij} v matici má hodnotu projektu, který právě počítáme a tento součet vydělíme počtem rozhodovačů). Dostaneme K průměrných hodnot, ty seřadíme vzestupně a máme výsledné pořadí projektů.

Pro sečtení pořadí pro všechny projekty musíme projít celou rozhodovací matici. Pro spočtení průměru z těchto K hodnot je musíme všechny projít a vydělit N. Hodnoty umíme setřídit v čase $N \log N$, neboť to jsou racionální čísla. Časová složitost je $O(NK + N + N \log N)$. Nešlo by to rychleji? V druhé fázi nemusíme hodnoty dělit číslem N, neboť nás ve výsledku bude zajímat pouze pořadí projektů. Tak dostaneme namísto racionálních čísel čísla přirozená a ta už umíme třídit v lineárním čase. Výsledná časová složitost je pak $O(NK)$. V programu je třídění zajištěno knihovní funkcí jazyka C#.

A jak je to s pamětí? Potřebujeme si uchovávat vstup a navíc i průměrné hodnoty pořadí projektů. Tedy je paměti je potřeba $O(NK + N)$.

Chody:	1	2	3	4
Průměrné pořadí:	2.33	1.33	3	3.33

Tabulka 1: Průměrné hodnoty pořadí chodů před setříděním.

Maximum agreement heuristic model (MAH)

Vytvoříme si frekvenční matici z rozhodovací matice ze zadání. Označme si dále součet prvků v i-tém řádku P_i a součet prvků v i-tém sloupci N_i. Vybereme prvek s nejmenším $P_i - N_i$, označíme ho jako nejméně preferovaný a odstraníme z rozhodovací matice. Přepočítáme celou matici a postup opakujeme, dokud nevyčerpáme všechny projekty.

$$\begin{pmatrix} 0 & 0 & 2 & 3 \\ 3 & 0 & 2 & 3 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 2 & 0 \end{pmatrix}$$

Obrázek 1: Frekvenční matice

Chody	1	2	3	4
P_i	5	8	3	2
N_i	4	1	6	7
$P_i - N_i$	1	7	-3	-5

Obrázek 2: Kompromis bude X-X-X-4.

Počítání frekvenční matice nám zabere $O(NK^2)$ operací. Pro výpočet hodnot P_i a N_i ji budeme potřebovat projít celou, tedy $O(K^2)$ operací. Najít pro každý projekt minimum z K prvků nám potrvá $O(K^2)$ času. Teď bychom potřebovali frekvenční matici aktualizovat, přepočítat hodnoty P_i, N_i a $P_i - N_i$ pro každý projekt. Ve skutečnosti nám stačí jen aktualizovat proměnnou $P_i - N_i$ pro všechna i, pro každý projekt. To nám potrvá $O(K^2)$. Výsledná asymptotická složitost algoritmu tedy bude $O(NK^2)$.

Potřebujeme si pamatovat kromě vstupu i frekvenční matici a pro každý projekt i proměnnou $P_i - N_i$, spotřebujeme $O(NK + K^2 + N)$ paměti.

Consensus ranking model (CRM)

Zavedeme si, co je to *preferenční matice* P pro permutaci K projektů, kterou značme p. Preferenční matice je rozměrů $K \times K$ a má na pozici $p(i), p(j)$ hodnotu $j - i$. Pro takovou matici platí $P_{k,l} = -P_{l,k}$, v programu stačí tedy počítat jen horní (čí dolní) trojúhelníkovou matici matice P, což nám ušetří paměť.

Zpátky k metodě CRM. V zadání nám každý rozhodovač udal pořadí projektů, ve kterém je preferuje. Na začátku algoritmu si tedy předpočteme pro každého rozhodovače $i \in [1, N]$ preferenční matici A^i pro jeho pořadí projektů.

$$\begin{pmatrix} 0 & -1 & 2 & 1 \\ 1 & 0 & 3 & 2 \\ -2 & -3 & 0 & -1 \\ -1 & -2 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & -1 & -2 & 1 \\ 1 & 0 & -1 & 2 \\ 2 & 1 & 0 & 3 \\ -1 & -2 & -3 & 0 \end{pmatrix} \quad \begin{pmatrix} 0 & -1 & 2 & 1 \\ 1 & 0 & 3 & 2 \\ -2 & -3 & 0 & -1 \\ -1 & -2 & 1 & 0 \end{pmatrix}$$

Obrázek 3: Preferenční matice každého rozhodovače pro pořadí projektů ze zadání

Poté generujeme všechny permutace K projektů. Pro každou permutaci si opět spočteme preferenční matici B.

$$\begin{pmatrix} 0 & -1 & 2 & 1 \\ 1 & 0 & 3 & 2 \\ -2 & -3 & 0 & -1 \\ -1 & -2 & 1 & 0 \end{pmatrix}$$

Obrázek 4: Preferenční matice B pro výslednou permutaci projektů (2 1 4 3)

Dále pro každou matici A^i vytvoříme matici vzdáleností S^i mezi maticemi A^i a B. Následně

spočteme součet s_i matice S^i a součet s pro danou permutaci jako součet všech s_i. Výsledný kompromis je permutace s nejmenším součtem s.

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 0 & 4 & 0 \\ 0 & 0 & 4 & 0 \\ 4 & 4 & 0 & 4 \\ 0 & 0 & 4 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$s_1 = 0$ $\qquad$ $s_2 = 24$ $\qquad$ $s_3 = 0$

Obrázek 5: Matice vzdáleností a jejich součty pro permutaci (2 1 4 3) s výsledným součtem $s = 24$

Jak dlouho to trvá? Na začátku si vytvoříme N preferenčních matic v čase $O(NK^2)$ a pak generujeme všechny permutace v $O(K!)$ a pro každou vytváříme preferenční matici v $O(K^2)$, počítáme vzdálenostní matice a součet permutace v $O(NK^2)$. Celková časová složitost je $O(NK^2+K!(K^2+NK^2)) = O(K!NK^2)$.

Jak je to s pamětí? Potřebujeme si pamatovat permutaci, s kterou pracujeme a dosud nejlepší permutaci ($O(K)$), preferenční matici A^i pro každého rozhodovače $O(NK^2)$ (stačí jen horní trojúhelníkové, do dolní trojúhelníkové můžeme schovat matici B). Matice vzdáleností si pamatovat nemusíme, stačí jen za běhu počítat součty a na ty nám konstantní paměť vystačí. Celkově tedy spotřebujeme $O(K + NK^2)$ paměti.

Hybrid distance-based model (DCM)

Nejprve spočteme frekvenční matici A pro aktuální rozhodovací matici. Poté generujeme všechny permutace K projektů. Pro každou permutaci vytvoříme rozhodovací matici tak, že každý řádek matice bude právě vygenerované pořadí projektů a z této matice spočteme frekvenční matici B. Nakonec spočteme součet vzdálenostní matice mezi A a B. Výsledné pořadí projektů je takové, které má tento součet minimální.

$$\begin{pmatrix} 0 & 0 & 2 & 3 \\ 3 & 0 & 2 & 3 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 2 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 0 & 3 & 3 \\ 3 & 0 & 3 & 3 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Frekvenční matice pro rozhodovací matici ze zadání

Frekvenční matice ideálního projektu (2 1 4 3)

Vzdálenostní matice se součtem 6

Jak je to s časem? Počítání frekvenční matice nám zabere $O(NK^2)$, generování permutací K projektů je nejpomalejší $O(K!)$. Spočítání ideální matice, vzdálenostní matice a jejího součtu trvá $O(K^2)$. Celkem tedy $O(K!K^2 + NK^2)$.

Pamatovat si stačí frekvenční a ideální matici a současnou permutaci. Spotřebujeme tedy $O(K^2 + K)$ paměti.

Přidávání nových algoritmů

Přidávání nových algoritmů do programu je snadné. Stačí jen vytvořit vlastní třídu v souboru *Algorithms.cs* a odvodit ji od třídy ConsensusAlgorithms. Vstup je k nalezení v proměnné *data* typu DataStorage. Výsledný kompromis přidaného algoritmu se ukládá do pole *ProjectOrder* a ten se pomocí metody AddResult třídy DataStorage uloží do proměnné *data*. Také je nutno přetížit metodu Start, která slouží ke spuštění algoritmu. Nakonec stačí v souboru *Settings.cs* přidat do třídy Settings informace o jméně algoritmu a přiřadit jméno i přidané třídě do proměnné *identifier*.

4 Závěrem

Možná zlepšení

Bylo by zajímavé se zamyslet nad jinými (vtipnějšími) implementacemi ohodnocovací funkce. Dále by bylo zajímavé zkusit vymyslet heuristiku pro pomalejší metody.

Výsledek motivačního příkladu

Návrhy jednotlivých metod na pořadí, ve kterém servírovat chody a body, které metody dostaly za kompromis.

	kompromis				body
BAK	2	1	3	4	5
MAH	2	3	1	4	4
CRM	2	1	4	3	8
DCM	2	1	4	3	8

Nejlépe si vedly metody CRM a DCM.

Testovací data

K dispozici jsou ve složce *tests* dvě sady testovacích dat. Data použitá v této dokumentaci a data z článku [1].

Zdroje

Veškeré informace byly čerpány z článku [1].

Reference

[1] *TAVANA, Madjid; LOPINTO, Frank; W. SMITHER, James. A Hybrid Distance-Based Ideal-Seeking Consensus Ranking Model. Journal of Applied Mathematics and Decision Sciences* [online]. 2007, Volume 2007, [cit. 2011-02-17]. Dostupný z WWW: <`http://downloads.hindawi.com/journals/ads/2007/020489.pdf`>.