KIV/PRO

Inkrementální algoritmus pro Role Mining problém

Fakulta aplikovaných věd

Jan Hinterholzinger

Úvod do problému

Představme si, že máme velkou organizaci, ve které potřebujeme spravovat oprávnění (například přístup do různých místností, použití tiskárny, použití výtahu atd.). Tyto oprávnění se nepřidělují jednotlivě jednotlivým uživatelům, ale oprávnění jsou seskupeny do rolí, které jsou následně přiřazovány uživatelům. To značně ulehčuje repetitivní přiřazení oprávnění uživatelům.

Pro naše přiřazení rolí k uživatelům (přiřazení role uživatelům a oprávnění rolím) tedy hledáme takové přiřazení rolí k uživatelům a oprávnění k rolím s nejmenším možným počtem rolí.

„Rozdělení rolí“ (Role Mining) problém je výpočetně těžký problém a přírůstková verze není jednodušší.

Definice elementů

Přiřazení role k uživatelům budeme reprezentovat maticí o velikosti m x k, kde m je počet uživatelů a k je počet rolí, která se skládá z binárních hodnot (0, 1). Matici tohoto přiřazení budeme označovat **UA**. Například tedy hodnota v poli $\mathbf{UA}_{ij}$ je pravdivá, pokud i-tý uživatel má přiřazenou j-tou roli.

Obdobně budeme reprezentovat přiřazení oprávnění k rolím. Matici označíme **PA** a bude velikosti k x n, kde k je počet rolí a n je počet oprávnění.

Matice reprezentující přímé přiřazení pravomocí k uživatelům je výsledkem vynásobení matice **UA** maticí **PA** a označíme ji **UPA**.

Zároveň si definujeme funkce users(**P**) a permissions(**U**), kde **P** je podmnožina všech oprávnění a U je podmnožina všech uživatelů. Funkce users(**P**) nám vrátí vektor (matice m x 1) s pravdivostními hodnotami uživatelů, zda mají všechny oprávnění z **P**. Obdobně permissions(**U**) nám vrátí vektor (matice 1 x n) s pravdivostními hodnotami oprávnění, zda všichni uživatelé **U** mají dané oprávnění.

Popis algoritmu

Algoritmus je rozdělen na dvě hlavní části. Přidání oprávnění a odebrání oprávnění.

Popis algoritmu přidávání

Vstup: Matice **UA**, **PA**, **UPA**, přidávaná dvojice <i, j>, kde $\mathbf{UPA}_{ij}$!= 1

Výstupem algoritmu jsou pozměněné matice **UA**, **PA** a **UPA**, tak že **UA** x **PA** = **UPA** a $\mathbf{UPA}_{ij}$ = 1

Algoritmus modifikuje matici **UPA**, tak aby hodnota prvku matice $\mathbf{UPA}_{ij}$ se rovnala 1. Poté vytvoří vektory **c** velikosti m x 1 a **d** velikosti 1 x n. Vektor **c** označuje (pravdivostní hodnotou 1) uživatele, kterému chceme přiřadit roli j. Vektor **d** označuje (pravdivostní hodnotou 1) oprávnění, které chceme přidělit uživateli i.

Následně iteruje přes všechny role s řídící proměnnou l, kde se testuje jeden ze tří stavů:

1) První z těchto stavů označuje situaci, kdy nově přidávané oprávnění lze přiřadit k již existující roli tedy, že l-tý sloupec matice **UA** je stejný jako vektor uživatelů, kteří mají všechny přidávané oprávnění. V takovém případě pozměníme l-tý řádek matice **PA** tak, že provedeme disjunkci s tímto řádkem a vektorem **d**. Pokud tedy všichni uživatelé v roli l mají oprávnění j, tak oprávnění j je přiřazeno k roli l. Může se stát, že ještě existuje jedna role p, která má stejná oprávnění jako uživatel i. Pokud taková opravdu existuje, tak se odstraní z matice **UA** a **PA**. Na konci tohoto stavu se spustí odebrání nadbytečných rolí, které mohly být v průběhu procedury vytvořeny, a ukončí se přidávání prvku.
2) Další stav, který může nastat je, že uživatel i má všechny oprávnění (permissions(**c**)), které jsou přiřazeny k roli l. Poté tedy l-tý sloupec matice **UA** změníme na výsledek disjunkce

daného sloupce a vektoru **c**. Tím přiřadíme uživatele i do role l. Nyní je potřeba zkontrolovat všechny ostatní role, zda neexistuje nějaká role, ke které jsou přiřazeni všichni uživatelé s oprávněním j. V případě nalezení této role se provede její odstranění z matice **UA** a **PA** (odstranění příslušných sloupce a řádku) a ukončíme cyklus. Opět odstraníme nadbytečné role, které mohly být při této proceduře vytvořeny a tím ukončujeme přidání spojení mezi hráčem a oprávněním.

3) Třetí případ posuzuje stav, kdy uživatelé přiřazení k l-té roli (l-tý sloupec matice **UA**) mají oprávnění j (v j-tém řádku matice **UPA**) a zároveň všechny oprávnění role l (l-tý řádek matice **PA**) jsou oprávnění uživatele i (i-tý řádek matice **UPA**). Tehdy se provede disjunkce přidělených uživatelů role l (l-tý sloupec matice **UA**) s vektorem **c** a uloží se do seznamu uživatelů role l (l-tý sloupec matice **UA**). a disjunkce přidělených pravomocí s rolí l (l-tý řádek matice **PA**) s vektorem **d** a uloží se do přidělených pravomocí do role l (l-tý řádek matice **PA**).

Pokud se v průběhu cyklu program nedostane ani do jednoho stavu, tak se přidá nový sloupec jako vektor **c** do matice **UA** a nový řádek jako vektor **d** do matice **PA**. Tím vytváříme novou roli, protože danou změnu oprávnění není možné provést při aktuálním počtu rolí.

Příklad přidání spojení:

Mějme matice:

$$\boldsymbol{UPA} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} = \boldsymbol{UA} \otimes \boldsymbol{PA} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

A chtějme přidat spojení <2,3> (2. uživateli přiřadit 3. oprávnění)

Nejprve modifikujeme matici **UPA** a vytvoříme vektory **c** a **d**.

$$\boldsymbol{UPA} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix} \boldsymbol{c} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \boldsymbol{d} = [0 \quad 0 \quad 1]$$

Vypočítáme si také výsledky funkcí permissions(**c**) a users(**d**).

$$permissions(\boldsymbol{c}) = [1 \quad 1 \quad 1] \;\; users(\boldsymbol{d}) = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

Projdeme nyní cyklem od l = 1 do l = k

Testujeme, zda l-tý sloupec **UA** je stejný jako vektor users(**d**) (l = 1)

$$\boldsymbol{UA}_{\_l} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

Vidíme, že users(**d**) se nerovná 1. sloupci matice **UA**, takže pokračujeme v algoritmu. Testujeme, zda i-tý řádek je stejný jako výsledek permissions(**c**).

$$\boldsymbol{PA}_{i\_} = [1 \quad 1 \quad 0]$$

Podmínka nevychází kladně ani zde, a tak přecházíme na testování třetího stavu.

$$\boldsymbol{UA}_{\_l} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \;\; \boldsymbol{PA}_{l\_} = [1 \quad 0 \quad 0]$$

Při kontrole třetího stavu zjišťujeme, že druhá část výrazu nevyhovuje. Končí první otočka cyklu a inkrementuje se řídící proměnná (l = 2). Nyní opět testujeme 1. případ nyní ale s druhým sloupcem.

$$\boldsymbol{UA}_{\_l} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

Druhý sloupec matice **UA** se shoduje s výsledkem funkce users(**d**), takže vstupujeme do první stavu. Změníme l-tý řádek matice PA tak, že se bude rovnat disjunkce sebe a vektoru d.

$$\boldsymbol{PA}_{l\_} = \boldsymbol{PA}_{l\_} \vee \boldsymbol{d} = [1 \quad 1 \quad 0] \vee [0 \quad 0 \quad 1] = [1 \quad 1 \quad 1]$$

Takto tedy bude vypadat upravená matice **PA**:

$$\boldsymbol{PA} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

Nyní projdeme všechny role kromě l-té a pokud se bude p-tý řádek matice **PA** shodovat s výsledkem funkce permissions(**c**), tak se odstraní role p z matic **UA** a **PA**. (p = 1)

$$\boldsymbol{PA}_{p\_} = [1 \quad 0 \quad 0]$$

P-tý řádek matice **PA** se neshoduje s výsledkem funkce permissions(**c**), takže se pokračuje další otočkou cyklu. (p = 3)

$$\boldsymbol{PA}_{p\_} = [1 \quad 1 \quad 1]$$

Tento řádek se shoduje s výsledkem funkce permissions(**c**), takže se tato role (3.) odstraňuje. Výsledné matice tedy budou vypadat následovně:

$$\boldsymbol{UA} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 0 \end{pmatrix} \boldsymbol{PA} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

Nyní máme korektně modifikované matice **UPA**, **UA** a **PA** a tímto se ukončuje přidání spojení <2,3>.

Popis algoritmu odebrání oprávnění

Vstup: Matice **UA**, **PA**, **UPA**, přidávaná dvojice <i, j>, kde $\mathbf{UPA}_{ij} = 1$

Výstupem algoritmu jsou pozměněné matice **UA**, **PA** a **UPA**, tak že **UA** x **PA** = UPA a $\mathbf{UPA}_{ij}$!= 1

Algoritmus zpočátku nastaví hodnotu na pozici i, j na 0 a vytvoří vektor **c** o velikosti m x 1 a vektor **d** o velikosti 1 x n. Ve vektoru **c** nastaví na pozici i hodnotu 1 a ve vektoru **d** nastaví na pozici j hodnotu 1.

Dále se iteruje přes všechny role (l -> k) s řídící proměnnou l.

Pokud dané propojení <i, j> je jediné propojení, které tvoří roli l, tak se odebere role l. (odebere se l-tý sloupec v matici **UA** a l-tý řádek v matici **PA**).

Pokud má role l pouze oprávnění vektoru **d** (jediné oprávnění j této role je členem odstraňované dvojice), tak se uživateli tato role odebere (v matici **UA** se prvek <i, l> nastaví na 0).

Pokud uživatelé mající roli l se shodují s vektorem **c** (uživatel odebírajícího spojení), tak se odstraní spojení mezi oprávnění j a rolí l (v matici **PA** se nastaví 0 na pozici <l, j>)

A nakonec pokud uživatel i má roli l a role l má přiřazené oprávnění j, tak se vytvoří nový vektor **b**, kam se uloží oprávnění role l bez oprávnění j (j-tý prvek se nastaví na 0). Dále se vytvoří vektor **a**, kam

se uloží uživatelé, kteří mají oprávnění jako uživatel z odstraňované dvojice (z vektoru **c**). Odstraníme propojení mezi uživatelem i a rolí l (nastavíme prvek matice **UA** na pozici <i, l> na 0). Námi vytvořené vektory **a** a **b** jsou novými rolemi a přidají se k našim maticím. Tedy vektor **a** jako nový sloupec matice **UA** a vektor **b** jako nový řádek matice **PA**.

Po ukončení cyklu se spustí odstranění nadbytečných rolí z **UA** a **PA** a ukončí se odebrání oprávnění.

Příklad odebrání spojení

Mějme matice:

$$\boldsymbol{UPA} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix} = \boldsymbol{UA} \otimes \boldsymbol{PA} = \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 0 \end{pmatrix} \otimes \begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

A chtějme odebrat spojení <2,3> (2. uživateli odebrat 3. oprávnění)

Nejprve modifikujeme matici **UPA** a vytvoříme vektory **c** a **d**.

$$\boldsymbol{UPA} = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \boldsymbol{c} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \boldsymbol{d} = [0 \quad 0 \quad 1]$$

Vypočítáme si také výsledky funkcí permissions(**c**) a users(**d**).

$$permissions(\boldsymbol{c}) = [1 \quad 1 \quad 0] \quad users(\boldsymbol{d}) = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$$

Nyní projdeme přes všechny role a testujeme, zda na následující podmínky. Testujeme, zda je dané propojení jediné, které tvoří roli l (l = 1).

$$\boldsymbol{UA}_{\_l} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} PA_{l\_} = [1 \quad 0 \quad 0]$$

Vidíme, že podmínka nebyla splněna, protože se zde nejedná o rovnost odebíraného spojení. Podmínka nebyla splněna a přecházíme na další dvě podmínky, které také nejsou splněné (první složená podmínka se skládá z těchto dvou jednoduchých). Nyní tedy zjišťujeme, zda uživatel i má roli l a zda role l má pravomoc j.

$$\boldsymbol{UA}_{il} = 1 \quad \boldsymbol{PA}_{lj} = 0$$

Role l neobsahuje pravomoc j, takže podmínka je nesplněna. Tato otočka cyklu končí a pokračuje se další (l = 2). Opět získáme l-tý sloupec matice **UA** a l-tý řádek matice **PA** a zjistíme, zda se jedná o jediné spojení, které tuto roli tvoří.

$$\boldsymbol{UA}_{\_l} = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \boldsymbol{PA}_{l\_} = [1 \quad 1 \quad 1]$$

Zjišťujeme, že se o jediné spojení tvořící tuto roli nejedná a že se nejedná ani o jedinou roli ve spojení s uživatelem nebo oprávněním z odstraňované dvojice. Proto pokračujeme na další podmínku ke zjištění, zda uživatel i má roli l a zda role l má pravomoc j.

$$\boldsymbol{UA}_{il} = 1 \quad \boldsymbol{PA}_{lj} = 1$$

Podmínka je splněna. Vytvoříme tedy vektory **a** a **b**, které budou novými sloupci a řádkami v maticích **UA** a **PA** a z matice **UA** se odebere (vynuluje) hodnota na pozici <i, l>.

$$\boldsymbol{a} = PA_{l\_} - j = [1 \quad 1 \quad 0]$$

$$\boldsymbol{b} = users\big(permissions(\boldsymbol{c})\big) = users([1 \quad 1 \quad 0]) = \begin{bmatrix}1\\1\\0\end{bmatrix}$$

$$\boldsymbol{UA} = \begin{bmatrix}1 & 1\\1 & 0\\1 & 0\end{bmatrix}$$

Přidáme tedy vektor **a** jako řádek do matice **PA** a vektor **b** jako sloupec do matice **UA**. Vzniknou nám následující modifikované matice **PA** a **UA**, které jsou společně s upravenou matici **UPA** výsledkem algoritmu pro odebírání spojení.

$$\boldsymbol{UA} = \begin{bmatrix}1 & 1 & 1\\1 & 0 & 1\\1 & 0 & 0\end{bmatrix} \quad \boldsymbol{PA} = \begin{bmatrix}1 & 0 & 0\\1 & 1 & 1\\1 & 1 & 0\end{bmatrix}$$

Popis algoritmu odstranění nadbytečných rolí

Oddělenou součástí tohoto algoritmu je i již několikrát zmíněné odstraňování nadbytečných rolí. Nadbytečné role poznáme tak, přidávají spojení mezi uživateli a oprávněními, které již existují. Takové role je možné odstranit bez dopadu na výsledek přiřazení uživatelů k rolím (matice **UPA**).

Iterujeme přes všechny role s řídící proměnou l (l = k; l → 1)

Uložíme si kopii přiřazení uživatelů k oprávnění (matice **UPA**) a vložíme ji do proměnné **CUPA**.

Procházíme všechny ostatní role s řídící proměnou p (p = k; p → 1) a odstraňujeme z matice **CUPA** takové prvky, které mají spojení s těmito rolemi. Pokud všechny spojení s rolí l jsou z matice **CUPA** odstraněny, tak role l může být odstraněna.

Časová náročnost algoritmu

Algoritmus přidání

V každém přidání prvku se musí vypočítat hodnoty funkcí users(d) a permissions(c). Tyto funkce mají lineární složitost a mohou být dokončeny v O(n) a O(m). Časová náročnost prvního případu přidání spojení je O(k * m) + O(k * m). Stejně takovou časovou složitost má i druhý případ, který se principiálně liší od prvního minimálně. Časová složitost třetího případu přidání je O(k * (m + n)). Smazání nadbytečných rolí, které zabere nejvíce časové náročnosti algoritmu přidání má složitost O(k * m * n).

Algoritmus odstranění spojení

U algoritmu přidání opět značně ovlivňuje časovou náročnost odstranění nadbytečných rolí, které má složitost O(k * m * n). Samotný algoritmus má složitost O(m * n), takže výsledná složitost se rovná O(k * m * n) + O(m * n).

Srovnání s ostatními algoritmy

Článek zmiňuje, standardní používání neinkrementální heuristický algoritmus, který je popsán v Ene et al. (2008) prezentovaný v Belohlavek and Vychodil (2010). S použitím dat z reálného světa byl inkrementální algoritmus výrazně rychlejší než neinkrementální. V případě inkrementálního

algoritmu byly časy potřebné na vložení a odstranění spojení značně nevyvážené. Naopak u neinkrementálního algoritmu jsou tyto časy více vyrovnané. Nejpomalejší část algoritmu je procedura na odstranění nadbytečných rolí. Tato část se dá v rámci přidávání nových pravomocí vynechat a tím bude tato část algoritmu ještě rychlejší. Bohužel v případě odebírání pravomocí nelze tuto část algoritmu přeskočit, protože odebírání vytváří nadbytečné role velmi často.

Závěr

Algoritmus považuji i přes náročnost na pochopení za zajímavý. Zajímala by mě nějaká praktická implementace tohoto algoritmu v nějakém softwaru, kde je možnost nastavovat oprávnění.
Z počátku jsem měl potíže se zorientovat v označování v původním textu, ale bylo vše pěkně vysvětlené v samostatné části. Program jsem se pokusil také implementovat v jazyce Java, abych algoritmus lépe pochopil.

Bibliografie

Trnecka, M., & Trneckova, M. (2020). An incremental algorithm for the role mining problem. Computers & Security, 94, 101830. https://doi.org/10.1016/j.cose.2020.101830 (https://www.sciencedirect.com/science/article/pii/S0167404820301036)

Role-based access control. Wikipedia, The Free Encyclopedia. [Online] 11. 10 2020. [Citace: 20. 10 2020.] https://en.wikipedia.org/wiki/Role-based_access_control.

Belohlavek, R., & Vychodil, V. (2010). Discovery of optimal factors in binary data via a novel method of matrix decomposition. Journal of Computer and System Sciences, 76(1), 3-20. https://doi.org/10.1016/j.jcss.2009.05.002 (https://www.sciencedirect.com/science/article/pii/S0022000009000415)