Hledání častých sekvencí

Návod ke čtvrtému cvičení

ZS 2011/2012

Úvod

V tomto cvičení si vyzkoušíte implementovat dva jednoduché algoritmy pro vyhledávání častých vzorů v sekvenčních datech. Existuje mnoho variant tohoto problému (jedna sekvence / několik sekvencí, různé definice frekvence atd.). My se omezíme na případ, kdy máme jednu sekvenci a frekvenci definujeme jako počet výskytů podsekvence vydělený maximálním možným počtem výskytů (což se pro podsekvenci (vzor) délky m s f výskyty v sekvenci S délky n rovná $\frac{f}{n-m+1}$).

Kostra algoritmu

Jako základ vám poslouží algoritmus pro vyhledávání častých podsekvencí s využitím kanonických kódů, s nímž jste se blíže seznámili na přednášce. Následující kód je uveden v obecné podobě, tj. není omezen pouze na hledání orientovaných, nebo neoriantovaných sekvencí. (Nelekejte se, že je kód nekomentovaný. Když si stáhnete data ke cvičení naleznete tam verzi s komentáři.) V matlabovském kódu budeme pracovat s přímým kódem pro sekvence - teorii týkající se kanonických kódů budeme využívat nepřímo jako návod pro rozšiřování častých podsekvencí.

```
function freqPatterns = aprioriFPM(data, sMin, initializeWords,...
    expandWords, findSupport, sortedAlphabet, prune)

freqPatterns = {};

sortedA = sortedAlphabet(data);
words0 = initializeWords(sortedA);

words = {};
```

```
for i = 1:size(words0,2)
    s = findSupport(words0{i}, data);
    words0{i}.support = s;
    if s ≥ sMin
        words{end+1} = words0{i};
    end
end

% we repeat this loop as long as new frequent patterns are found
while ¬isempty(words)
    for i = 1:size(words,2)
        freqPatterns{end+1} = words{i};
    end
    wordCand = expandWords(words,sortedA)
    % checks if all subsequences are frequent (i.e. present in words)
    wordCand = prune(words, wordCand);

    words = {};
    for i = 1:length(wordCand)
        s = findSupport(wordCand{i},data);
        wordCand{i}.support = s;
        if s≥sMin
            words{end+1} = wordCand{i};
        end
    end
end
```

Co máte doplnit...

Vaším úkolem bude doimplementovat do m-souborů *directedSequenceMining.m* a *undirectedSequenceMining.m* funkce *expandSequences(oldPatterns, alphabet)*.

```
function frequentPatterns = directedSequenceMining(data, sMin)

frequentPatterns = aprioriFPM(data, sMin, @initializeWords,...
    @expandSequences, @findSupport, @sortAlphabet, @prune);

function emptyWordArray = initializeWords(sortedAlphabet)
emptyWord.sequence = {};
emptyWordArray = {emptyWord};

function nextPatterns = expandSequences(oldPatterns, alphabet)
% zacatek kodu, ktery studenti mit nebudou

% konec kodu, ktery studenti mit nebudou
```

```
function support = findSupport(pattern, data)
% ...

function sortedAlphabet = sortAlphabet(data)
sortedAlphabet = sort(unique(data.sequence));

function filtered = prune(shorter, longer)
% ...
```

V případě orientovaných sekvencí očekává funkce *expandSequences(oldPatterns, alphabet)* na svém vstupu *cell array* obsahující časté podsekvence délky n a abecedu používaných znaků *alphabet* (kterou vám vytvoří připravená funkce *sortAlphabet(data)*) a vrací podsekvence délky $n+1$. Následující kousek kódu naznačuje, jak lze v Matlabu vytvářet nové podsekvence z kratších podsekvencí

`newPattern.sequence = [oldPattern.sequence { 'a'} ].`

Funkci *expandSequences(oldPatterns, alphabet)* budete implementovat dvakrát - jednou pro hledání častých vzorů v orientovaných sekvencích (tj. těch, které mají definovaný *směr*) a jednou v neorientovaných sekvencích. Kód a popis uvedený výše odpovídá verzi s orientovanými sekvencemi, zatímco kód níže odpovídá verzi s neorientovanými sekvencemi.

```
function frequentPatterns = undirectedSequenceMining(data, sMin)

frequentPatterns = aprioriFPM(data, sMin, @initializeWords,...
    @expandSequences, @findSupport, @sortAlphabet, @prune);

function initial = initializeWords(sortedAlphabet)

emptyWord.sequence = {};
emptyWord.symmetryFlag = 1;
initial = {emptyWord};
for i = 1:size(sortedAlphabet,2)
    iw.sequence = {sortedAlphabet{i}};
    iw.symmetryFlag = 1;
    initial{end+1} = iw;
end

function nextPatterns = expandSequences(oldPatterns, alphabet)
% zacatek kodu, ktery studenti mit nebudou

% konec kodu, ktery studenti mit nebudou
```

```
function support = findSupport(pattern, data)
% ...

function sortedAlphabet = sortAlphabet(data)
sortedAlphabet = sort(unique(data.sequence));

function filtered = prune(shorter, longer)
% ...
```

Můžete si všimnout, že se od sebe funkce *initializeWords* v případě orientovaných a neorientovaných sekvencí poněkud liší. Hlavní rozdíly jsou dány různým tvarem kanonických kódů pro orientované a neorientované sekvence. Prvním z těchto rozdílů je to, že v případě neorientovaných sekvencí pro každou vygenerovanou podsekvenci udržujeme atribut *symmetryFlag*, který má hodnotu 1, pokud je daná podsekvence palindromem, a jinak má hodnotu 0. Druhý rozdíl spočívá v tom, že hned na začátku vygenerujeme podsekvence délky nula[1], což děláme i v případě orientovaných sekvencí, ale nyní kromě toho vygenerujeme i podsekvence délky 1. Důvodem je to, že u neorientovaných sekvencí vytváříme nové podsekvence přidáním dvou písmen - jednoho na začátek a jednoho na konec (tj. vytváříme současně podsekvence délky $n+1$ a $n+2$ z podsekvencí délky $n-1$ a n).

Jak to máte doplnit...

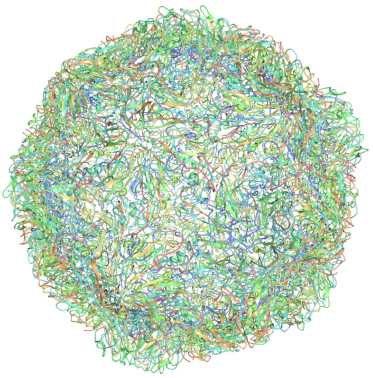
... jste se mohli dozvědět na přednášce. Tam jste se dozvěděli, jak vypadají kanonické kódy pro orientované a neorientované sekvence. Kromě toho jste se také dozvěděli, jak lze využít těchto kanonických kódů pro efektivní generování podsekvencí. Na konci tohoto dokumentu najdete několik tipů, jak pracovat v Matlabu s tzv. *cell arrays* a se strukturami, což by se vám, pokud jste se s tím ještě nesetkali, mohlo při řešení tohoto úkolu hodit.

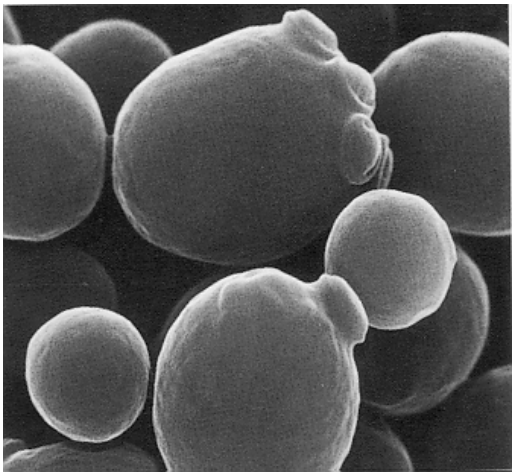
Testovací data (sekvence DNA viru a kvasinky)

Implementované algoritmy použijete pro hledání častých sekvencí v DNA viru *Equine rhinitis A* (Obrázek 1) a kvasinky *Saccharomyces cerevisiae* (konkrétně jejího chromozomu číslo III). Pro testování při implementaci používejte raději sekvenci viru, protože je kratší a nebude tak dlouho trvat, než dostanete výsledek (nebo objevíte chybu).

Data načtete pomocí příkazu *load_dna*, který nahraje sekvenci viru do proměnné *virus_dna* a sekvenci kvasinky do proměnné *saccharomyces_dna*.

[1] Pro účely tohoto cvičení považujeme prázdnou podsekvenci za korektní podsekvenci.





Obrázek 1: *Vlevo:* Trojrozměrná struktura proteinové obálky viru *Equine rhinitis A*. *Vpravo:* Kvasinky *Saccharomyces cerevisiae*.

Níže máte uvedenu ukázku použití hotových algoritmů na sekvenci viru.

```
load_dna;
dna_data.sequence = virus_dna;

%% directed sequence mining

seqs = directedSequenceMining(dna_data, 0.01);
fprintf('Printing results of directed sequence mining...\n');
printResults(seqs, 3);
```

Pokuste se diskutovat rozdíly mezi častými vzory pro viry a kvasinky, což jsou eukaryotní organismy. (*Může se vám hodit: komplementární báze A-T, C-G*).

Očekávaný výsledek

Pro kompletní genom viru a minimální frekvenci 0.01 byste měli nalézt následující čtyři orientované sekvence: *attt*, *ttga*, *tttg*, *tttt*. Podbně byste pro genom viru měli nalézt následující dvě neorientované sekvence: *gttt*, *tttt*.

Pár užitečných příkazů pro práci se strukturami a "cell arrays"v Matlabu

Pokud jste se v Matlabu ještě nesetkali se strukturami a cell-arrays, můžete najít několik užitečných příkazů v této sekci.

Cell arrays

Cell arrays (dále jim budeme říkat jen *pole*) se od matic odlišují především tím, že mohou obsahovat nejen čísla, ale i textové řetězce nebo struktury nebo vnořená další pole. Prázdné pole můžeme vytvořit například takto `a = {};`, což je velice podobné tomu, jak v Matlabu vytváříme prázdné vektory/matice.

K prvkům pole přistupujeme takto: `a = {'a', 'b', 'c'}; x = a{1}; % x = 'a'`. Všimněte si, že narozdíl od matic se zde používají složené závorky.

Dále budete nejspíš potřebovat spojovat pole. To lze provést následovně: `a = {'a','b'}; b = {'c','d'}; merged = [a b];`. Za povšimnutí stojí, co by se stalo, pokud bychom místo hranatých závorek použili závorky složené, tj. `merged = {a b};`. V takovém případě bychom nedostali pole se čtyřmi prvky, ale pole obsahující dvě vnořená pole.

Struktury

Struktury v Matlabu jsou velice flexibilní. Například

```
data.sequence = {'a','b','c'};
```

vytvoří strukturu obsahující jednu položku s názvem *sequence* typu *cell array* (není nijak potřeba strukturu *data* dopředu inicializovat). Tímto způsobem je možné přidat libovolný počet dalších položek. K obsahu potom přistupujeme jednoduše, například $x = data.sequence$.