Tomáš Oberhuber

Faculty of Nuclear Sciences and Physical Engineering
Czech Technical University in Prague

Video na Youtube

Hledání motivů

Definition 1

Motivem budeme rozumět určitý vzor dané délky, který se často opakuje v zadaném řetězci.

Budeme řešit následující úlohu:

Máme t řetězců $r_1, \ldots, r_t$, každý má délku n. V každém řetězci chceme najít podřetězec délky l tak, aby si všech t vybraných podřetězců bylo co nejpodobnějších. Této úloze říkáme hledání motivů.

Hledání motivů

Example 2

.GA**ACTCATGG**TG
.AA**AAGCACGG**TC
TCA**AAGCAAGG**C
CCT**AATCAGGG**C
...**AAGTATGG**ACTC
ACT**AAGCAGGG**T
...**TCTCACGG**CCCA
...**CCTCGTGG**TGGG
.TA**CCGTATGG**TT
ACC**ACTCGTCG**A

Hledání motivů

- ve výkladu použijeme terminologii z bioinformatiky a řetězce délky l budeme nazývat l-mery (= polymery délky l)
- v každém řetězci chceme najít jeden l-mer tak, aby si l-mery v různých řetězců byly co nejvíce podobné
- každý l-mer je jednoznačně určen svou pozicí s_i v i-tém řetězci, kde

$$1 \leq s_i \leq n - l + 1$$

- všech t l-merů lze jednoznačně určit vektorem

$$\vec{s} = (s_1, \ldots, s_t)$$

Hledání motivů

- naším cílem nyní bude najít způsob, jak ohodnotit podobnost S mezi l-mery $\vec{s}$ čímž úlohu převedeme na problém

$$\vec{s}^* = \arg\max_{\forall \vec{s}} S(\vec{s}).$$

- k tomu použijeme tzv. **matici zarovnání** (*alignment matrix*) $A(\vec{s})$
- to jsou všechny l-mery zapsané pod sebou, takže jde o matici o t-řádcích a l sloupcích
- z matice zarovnání odvodíme **profilovou matici** $P(\vec{s})$
- ta udává, kolikrát se daný znak vyskytuje v určitém sloupci
- podle profilové matice vybereme konsensus $C(\vec{s})$, tj. v každém sloupci znak, který má nejvíce výskytů
- nakonec podle počtu výskytu jednotlivých znaků v konsensu napočítáme celkové skóre shody $S(\vec{s})$

Hledání motivů

Example 3

Mějme jednu pevně danou t-tici l-merů pro $t = 7, l = 8$

		A	T	C	C	A	G	C	T
		G	G	G	C	A	A	C	T
		A	T	G	G	A	T	C	T
$A(\vec{s})$		A	A	G	C	A	A	C	C
		T	T	G	G	A	A	C	T
		A	T	G	G	C	A	T	T
		A	T	G	G	C	A	C	T
	A	5	1	0	0	5	5	0	0
$P(\vec{s})$	T	1	5	0	0	0	1	1	6
	G	1	1	6	3	0	1	0	0
	C	0	0	1	4	2	0	6	1
$C(\vec{s})$		A	T	G	C	A	A	C	T
$\max_{i=1,\ldots,t} P(\vec{s})_i$.		5	5	6	4	5	5	6	6

$$S(\vec{s}) = 5 + 5 + 6 + 4 + 5 + 5 + 6 + 6 = 42$$

Hledání motivů

- nyní již stačí najít $\vec{s}^* = \arg\max_{\vec{s}} S(\vec{s})$
- platí

$$\frac{lt}{4} \leq S(\vec{s}) \leq lt$$

- maximum tedy existuje

Kombinatorické optimalizace

- tím jsme úlohu převedli na úlohu z kombinatorické optimalizace (*combinatorial optimization*)
- obecně musíme generovat všechny možné řetězce z určitého stavového stromu
- při hledání motivu generujeme všechny možné vektory $\vec{s}$ o délce t a na každé pozici může být číslo 1 až $n - l + 1$, tj. $(n - l + 1)^t$ vektorů
- obecně chceme generovat všechny možné řetězce o délce L tvořené abecedou o k znacích

Kombinatorické optimalizace

```
1:  procedure ALLLEAVES( L, k)
2:      a:=(1,...,1)
3:      while true do
4:          vypiš a
5:          a:=NextLeaf( a, L, k )
6:          if a = (1,...,1) then
7:              return
8:          end if
9:      end while
10: end procedure
```

```
1:  procedure NEXTLEAF( a, L, k)
2:      for  i = L to 1 do
3:          if a_i<k then
4:              a_i := a_i + 1
5:              return
6:          end if
7:          a_i := 1
8:      end for
9:      return a
10: end procedure
```

Kombinatorické optimalizace

- takto lze generovat celou množinu řetězců, přes kterou hledáme optimální řešení
- výpočet lze urychlit technikou zvanou *branch-and-bound*
- generování řetězců si zorganizujeme do k-árního stromu, ve kterém na n-té úrovni máme fixováno prvních $n-1$-znaků řetězce a zbytek je neznámý
- pro každou větev pak uděláme odhad, jaké nejlepší skóre v ní můžeme dosáhnout
- podle počtu nezafixovaných znaků potřebujeme odhadnout, o kolik maximálně se může změnit optimalizovaná funkce

Kombinatorické optimalizace

- hledání motivu
  - generujeme postupně všechny možné složky vektoru $\vec{s} = (s_1, \ldots, s_t)$
  - pokud je m posledních složek nezafixovaných, můžeme je teoreticky nastavit tak, že budou ukazovat na úplně stejné podřetězce řetězců $r_1, \ldots r_t$
  - mohu tak maximálně získat lm shod, víc ale ne
  - pokud je tedy nejlepší doposud dosažené skóre větší o více než lm v porovnání se současným skóre, nemá smysl za současného stavu hledat dál a celou větev stromu lze přeskočit

Kombinatorické optimalizace

- procházení stavového stromu upravíme na procházení jeho vrcholů, ne pouze listů
- i udává úroveň ve stromu

```
1:  procedure NextVertex( a, i, L, k)
2:      if  i < L  then // posun o úroveň níž
3:          a_{i+1} := 1
4:          return (a, i + 1)
5:      else
6:          for  j := L to 1 do // dokud je a_j = k vracíme se
    výše
7:              if  a_j < k then
8:                  a_j := a_j + 1
9:                  return ( a, j )
10:             end if
11:         end for
12:     end if
13:     return ( a, 0 )
14: end procedure
```

Kombinatorické optimalizace

- a dále potřebujeme proceduru pro přeskočení větve stavového stromu

```
1: procedure BYPASS( a, i, L, k)
2:     for j := i to 1 do // dokud je a_j = k vracíme se výše
3:         if a_j < k then
4:             a_j := a_j + 1
5:             return ( a, j )
6:         end if
7:     end for
8:     return ( a, 0 )
9: end procedure
```

Kombinatorické optimalizace

- nyní můžeme napsat samotný algoritmus pro hledání motivů

```
1: procedure BranchAndBoundMotifSearch(r_1, ..., r_t, t, n, l)
2:     s⃗ := (1, 1, ..., 1)
3:     bestScore := 0
4:     i := 1
5:     while i > 0 do
6:         if i<t then
7:             optimisticScore := Score(s⃗, i, r_1, ..., r_t) + (t − i)l
8:             if optimisticScore < bestScore then
9:                 (s⃗, i) := Bypass(s⃗, i, t, n − l + 1)
10:            else
11:                (s⃗, i) := NextVertex(s⃗, i, t, n − l + 1)
12:            end if
13:        else
14:            if Score(s⃗, r_1, ..., r_t) > bestScore then
15:                bestScore := Score(s⃗, r_1, ..., r_t)
16:                bestMotif := s⃗
17:                (s⃗, i) := NextVertex(s⃗, i, t, n − l + 1)
18:            end if
19:        end if
20:    end while
21:    return bestMotif
22: end procedure
```

Hladový algoritmus pro hledání motivů

- jde pouze o aproximační algoritmus, u kterého neznáme aproximační poměr
- v praxi ale dává dobré výsledky a funguje rychle
- algoritmus je součástí programu Consensus
  - Gary Stormo, Gerald Hertz, 1989

Gary Stormo

Hladový algoritmus pro hledání motivů

- algoritmus nejprve vezme první dva řetězce r_1, r_2 a najde v nich dva nejbližší l-mery pomocí Hammingovy vzdálenosti
- následně v každé z t zbývajících sekvencí najde nejbližší motiv
- program Consensus navíc v první fázi vybere více kandidátů na motiv, řádově 1000

Hladový algoritmus pro hledání motivů

```
1: procedure GreedyMotifSearch(r_1, ..., r_t, t, n, l)
2:     bestMotif := (1, 1, ..., 1)
3:     s⃗ := (1, 1 ..., 1)
4:     for s_1 := 1 to n − l + 1 do
5:         for s_2 := 1 to n − l + 1 do
6:             if
   Score(s⃗, 2, r_1, ..., r_t) > Score(bestMotif, 2, r_1, ..., r_t)
   then
7:                 bestMotif_1 := s_1
8:                 bestMotif_2 := s_2
9:             end if
10:        end for
11:    end for
12:    s_1 := bestMotif_1
13:    s_2 := bestMotif_2
14:    for i := 3 to t do
15:        for s_i := 1 to n − l + 1 do
16:            if
   Score(s⃗, i, r_1, ..., r_t) > Score(bestMotif, i, r_1, ..., r_t)
   then
17:                bestMotif_i := s_i
18:            end if
19:        end for
20:        s_i := bestMotif_i
21:    end for
22:    return bestMotif
23: end procedure
```