Hledání motivů

Projection vs Pattern branching

Challenge Problem

- Najít motif ve vzorku
  - 20 “náhodných” sekvencí (e.g. 600 bp)
  - každá sekvence má implantován pattern délky 15
  - každý pattern se vyskytuje s nejvýše 4 chybami jako(15,4)-motif.

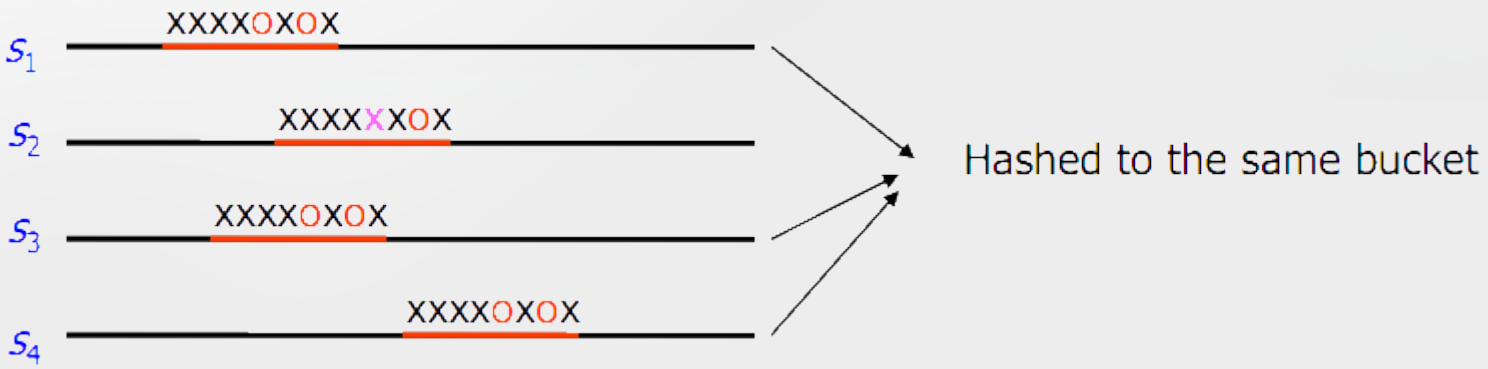
Implantování motivu **AAAAAAGGGGGGG** se 4 mutacemi

Některé termíny

- M – motiv
- t – počet DNA sekvencí ve vzorku
- n – délka každé sekvence DNA
- DNA – vzorek DNA sekvencí (pole t x n)
- l – délka motivu (l-mer)
- s_i – počáteční pozice l-mer v sekvecni I
- $s=(s_1, s_2, \ldots, s_t)$ – pole počátečních pozic motivu

Projekční algoritmus

Vybereme k pozic z l náhodně, ty pak použijeme pro každý l-mer x jako hash h(x). Pokud má dostatečný počet l-merů stejný hash, dá se očekávat, že budou blízké vloženému motivu.



Projekční algoritmus

- Vybereme ***k*** pozic z *l* náhodně. Hash l-meru x je pak zřetězení těchto *k* pozic x.
- Hlavní myšlenka:
  - Pokud $k < l - d$, pak je velká šance, že několik z *t* instancí vloženého motivu spadne do stejného pole hashovací tabulky.
  - Budeme se snažit získat motiv z polí hashovací tabulky, která obsahují dostatek l-merů, u těchto je velká šance, že z nich budeme moci odvodit motiv.

Projekční algoritmus

- Výběr pole hashovací tabulky obsahující motiv:
  - Algoritmus neví, kde je zahashován motiv
  - Snaží se získat motiv ze všech polí, která obsahují alespoň ***s*** l-merů
- První část algoritmu tedy je heuristika pro nalezení slibných l-merů.
- Algoritmus má tři parametry:
  - Projekční velikost ***k***
  - Práh ***s*** pro výběr polí hashovací tabulky
  - Počet nezávislých cest ***m***

Projekční algoritmus

- Projekční algoritmus by měl zahashovat dostatek instancí motivu do jednoho pole, zároveň by se měl vyhnout kontaminaci náhodnými l-mery.
- Jak vybrat ***k***?
  - Hashujeme $t\,(n - l + 1)$ l-merů do 4^k polí.
  - Pokud zvolíme k t.ž. $\boldsymbol{4^k > t\,(n - l + 1)}$ tak průměrné pole hashovací tabulky bude obsahovat nejvýše 1 l-mer

Projekční algoritmus

- Jak vybrat ***s***?
- Pro challenge problém je rozumné ***s = 3*** nebo ***4***, neočekáváme totiž, že by se příliš instancí zahashovalo do stejného pole.

Projekční algoritmus

- Jak vybrat počet nezávislých cest ***m***?
- Chceme míru spolehlivosti ***q = 0.95***, že hashovací tabulka bude obsahovat alespoň ***s*** instancí motivu v jednom políčku při alespoň jednom průchodu.
- Pravděpodobnost že se instance motivu zahashuje do správného políčka:
  -

$$p(l,d,k) = \frac{\binom{l-d}{k}}{\binom{l}{k}}$$

Projekční algoritmus

- Pravděpodobnost, že se méně než s instancí zahashuje do správného políčka:

$$B_{t,p(l,d,k)(s)} = \sum_{i=0}^{s} \binom{t}{i} p^i (1-p)^{t-i}$$

- Pokud je algoritmus spuštěn *m*-krát, tak pravděpodobnost, že s nebo více instancí motivu bude zahashováno do stejného políčka je:

$$1 - (B_{t,p(l,d,k)(s)})^m \geq q$$

Projekční algoritmus

- Zvolíme tedy m následovně:

$$m = \left\lceil \frac{\log(1-q)}{\log(B_{t,p(l,d,k)(s)})} \right\rceil$$

- Užitím tohoto vzorce pro *m* a s dříve zvolenými parametry *k* a *s* jsou potřeba nejvýše tisíce nezávislých cest pro nalezení motivu (většinou o hodně méně).

Projekční algoritmus

- Pseudokód:
    1. Vyber náhodně *k* pozic z *l*.
    2. Zahashuj všechny l-mery daných sekvencí.
    3. Prozkoumej všechny pole, do kterých se zahashovalo více než *s* l-merů a zlepši nalezené motivy.
    4. Opakuj *m*-krát, vrať motiv s nejlepším skóre.

Projekční algoritmus

- Jak zlepšit nalezený motiv?
- Nalezené motivy zlepšíme pomocí algoritmu **expectation maximization (EM)**
- Založeno na předpokladu, že instance motivu jsou generovány podle pravděpodobnostního modelu, který je odlišný od pravděpodobnosti výskytu bází v DNA.
- Algoritmus střídá iterativně 2 fáze:
  - Expectation fáze – počítáme očekáváný model motivu
  - Maximization – snažíme se najít l-mery, které modelu nejlépe odpovídají

Projekční algoritmus

- Nakonec nám vyjde množina l-merů T nejlépe odpovídajících nalezenému modelu.
- Vložený motiv pak bude l-mer, jehož součet Hammingovských vzdáleností od T je nejmenší přes všechny iterace.

Pattern Branching algoritmus

- **Finding subtle motifs by branching from sample strings**, Alkes Price, Sriram Ramabhadran and Pavel A. Pevzner, 2003
- Při hledání motivu zkoumáme patterny A_k s hammingovou vzdáleností **h(M, A_k) = k**
- Těch je ovšem

$$\binom{a}{b} 3^k$$

Pattern Branching algoritmus

- Není potřeba počítat h pro všechny
- Stačí zkonstruovat cestu

  $\mathbf{A_0 \rightarrow A_1 \rightarrow \ldots A_k}$

- postupnou aplikací funkce `BestNeighbour()`, která přiřadí patternu A jeho souseda ve hammingove vzdálenosti h = 1 s nejlepším skóre

Pattern Branching algoritmus

Problémy:

- Jak definovat skóre?

- Jak definovat `BestNeighbour(A)`?

Pattern Branching algoritmus

- Jak definovat skóre?

$$d(A, S_i) = min\{ d(A, P) : P \subset S_i, P\ je\ l-mer \}$$

$$d(A, S) = \sum_{S_i \in S} d(A, S_i)$$

Pattern Branching algoritmus

- Jak definovat `BestNeighbour(A)`?
- Jako vzorek B s h(A, B) = 1 s nejmenší celkovou vzdáleností h(B,S)

Pattern Branching algoritmus

- Výsledný algoritmus

```
PatternBranching(S, l, k)
        Motif ← arbitrary motif pattern
        For each l-mer A_0 in S
                For j ← 0 to k
                        If d(A_j , S ) < d(Motif, S )
                        Motif ← A_j
                        A_j+1 ← BestNeighbor(A_j)
        Output Motif
End.
```

Pattern Branching algoritmus

- Poznámky a vylepšení:
- Je možné udržovat množinu nejlepších motivů – podrobnější prohledávání
- Výpočet $\mathbf{d(A_0,S)}$ je možné urychlit sdílením mezi sousedními $\mathbf{A_0}$
- Výpočet nejlepšího souseda lze urychit nahrazením $\mathbf{S_i}$ za **P**

Porovnání algoritmů

n = 600 t = 20	**Projection alg.**		**Pattern branching**	
	Úspěšnost	Čas [s]	Úspěšnost	Čas [s]
(10, 2)	0.92	75.7	Okolo 100%	653
(11, 2)	0.96	114	Okolo 100%	660
(13, 3)	0.98	128	Okolo 100%	658
(15, 4)	0.98	150	0.968	650

Závěr

- Projekční algoritmus značně urychlí výběr kandidátů pro motiv, dává dobré výsledky i při snížení počtu iterací ve prospěch rychlosti.
- Pattern branching by teoreticky měl být rychlejší, nám se ovšem nepovedlo provést všechny optimalizace, abychom dovedli rychleji řešit challenge problém.