Humboldt-Universität zu Berlin
Institut für Informatik

Übung
Algorithmische Bioinformatik

Aufgabenblatt 2

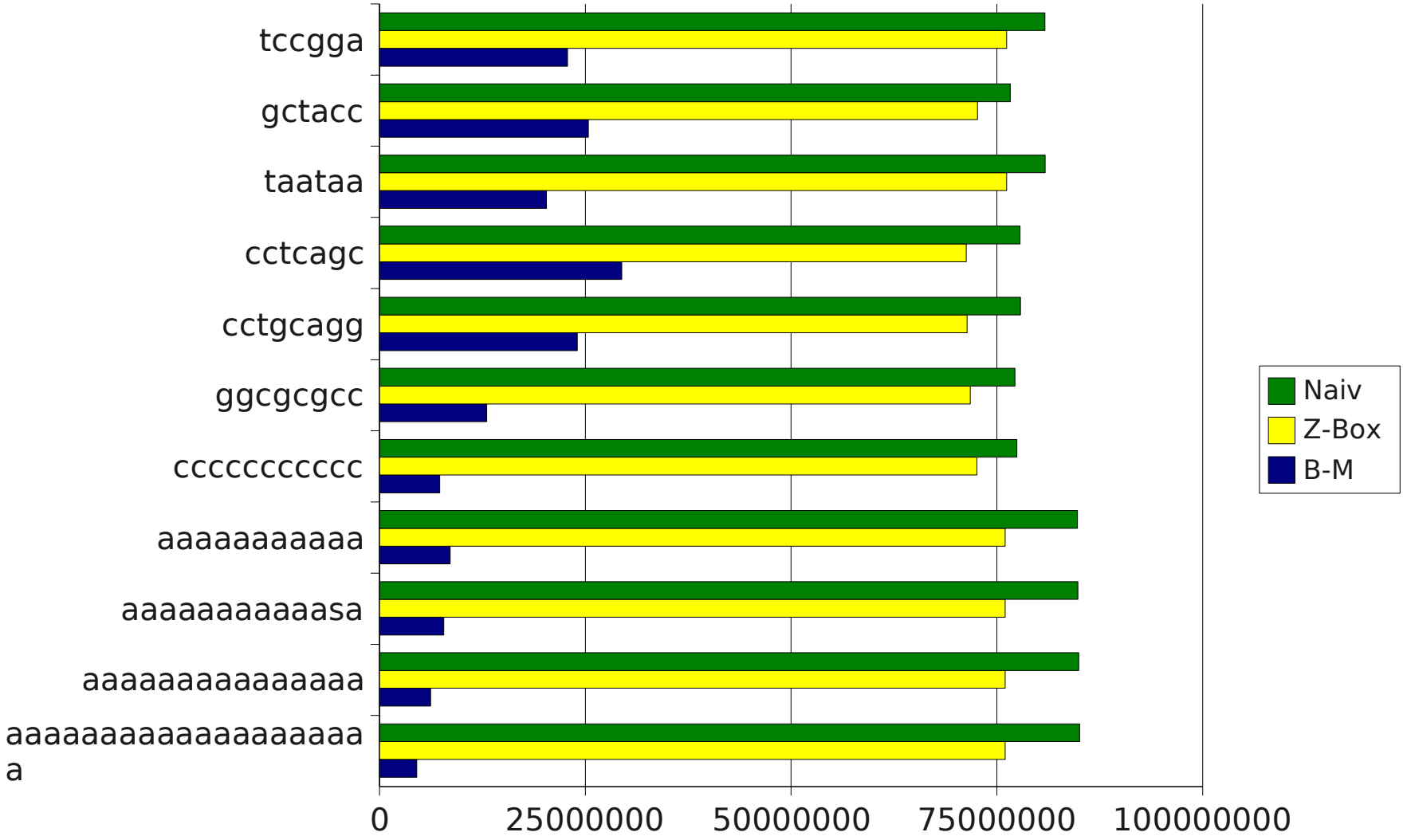
Gruppe 2:
Martin Stigge
Alexandra Rostin

2007-11-29

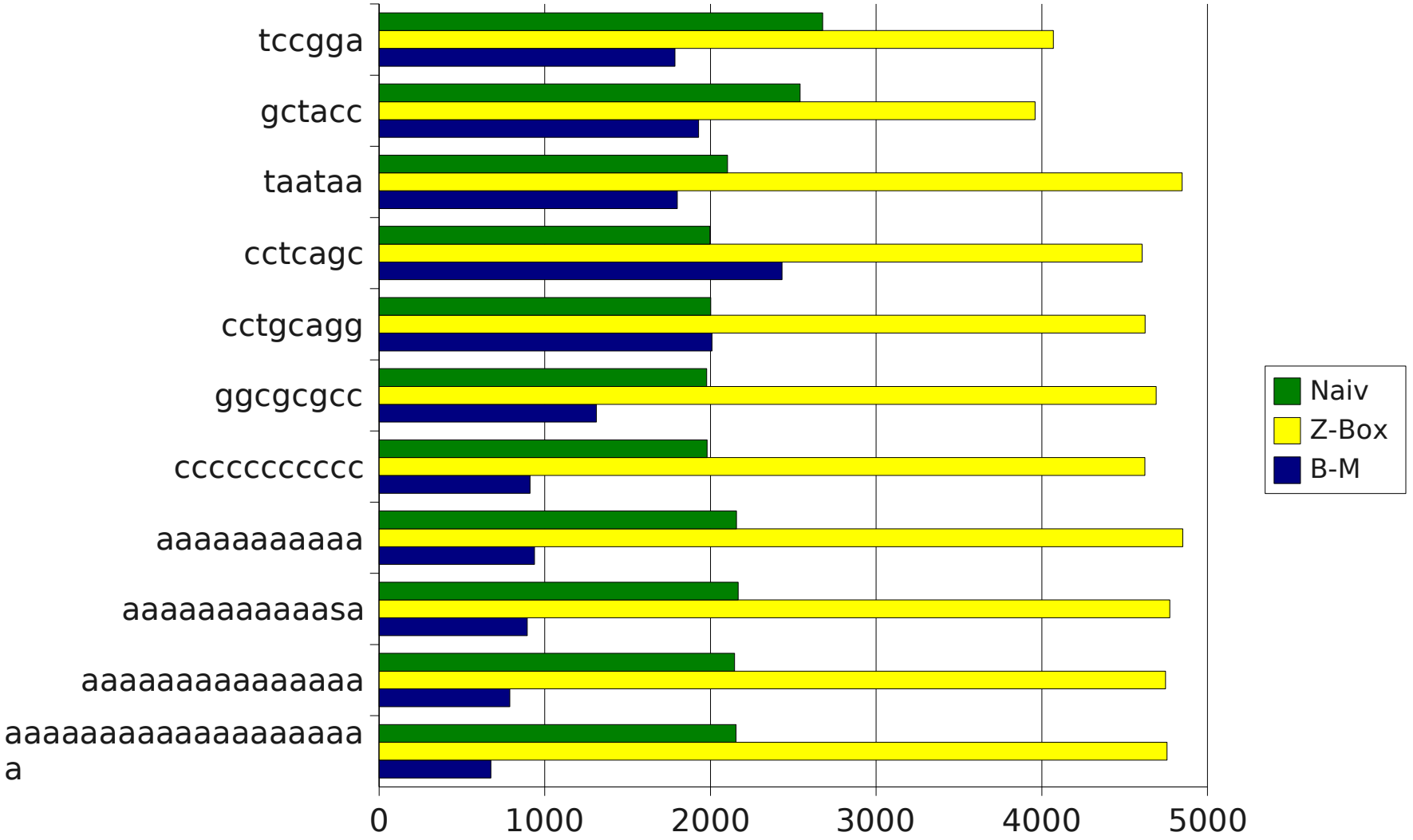
Aufgabe 1 - Boyer-Moore

- Implementation Boyer-Moore
- Größte Hürde, mehr noch als bei Z-Box:
  - “Differenz” in Folie → “Differenz” in Java-Code
  - “Position” in Folie → “Position-1” in Java-Code

Zeichenvergleiche

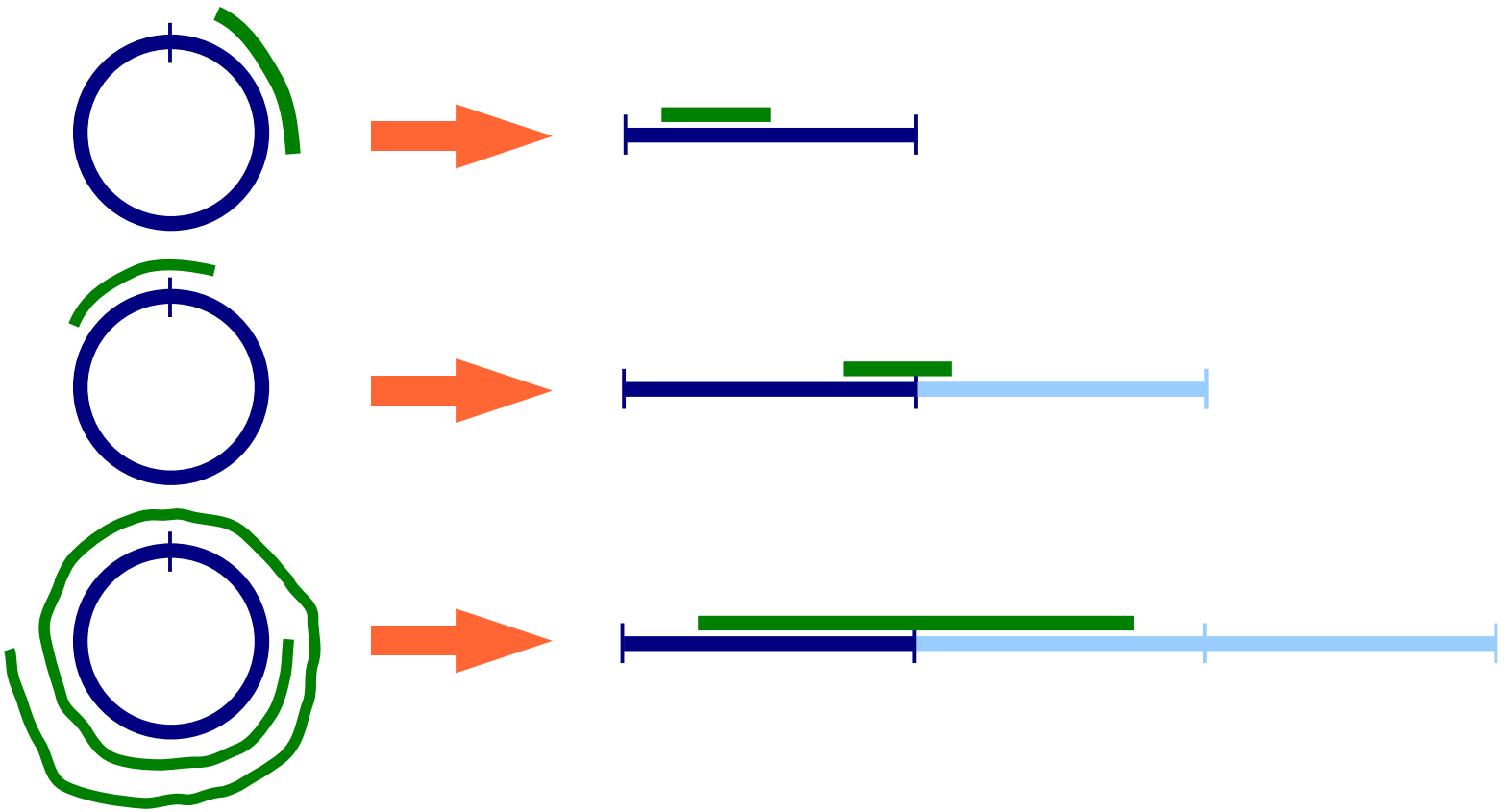


Laufzeiten in ms auf Amsel

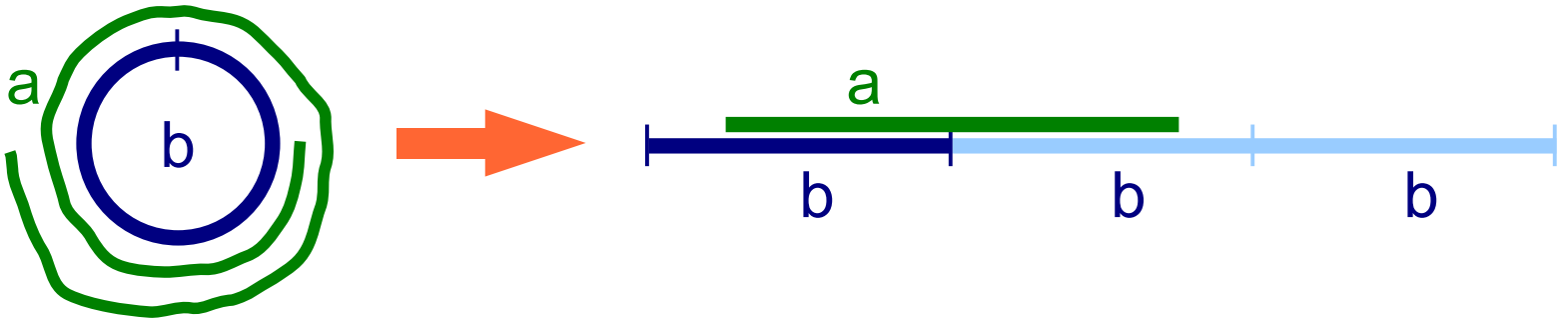


Aufgabe 2 - zirkuläre Strings

- Erstmal Teil 2: Substring eines zirkulären Strings
- Idee: Template “abrollen” und darauf normale Suche:



Aufgabe 2 - zirkuläre Strings



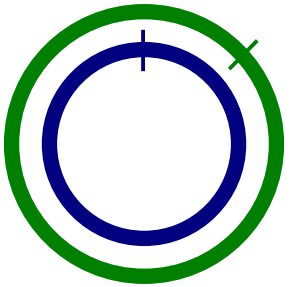
- Also: Auf “bekannten” Fall zurückführen, und dabei
- eine bekannt lineare Suche `LinSearch(P, T)` benutzen:

```
                    1. T := b;
CircSearch(a, b):   2. while |T| < |a|+|b|-1:
                    3.       T := T + b;
                    4. return LinSearch(a, T)
```

- Verlängern von T dauert O(|a|+|b|), LinSearch() ebenso

Aufgabe 2 - zirkuläre Strings

- Jetzt Teil 1: Zirkuläre Rotation eines Strings
- Nutze Teil 2:
  - a ist zirkuläre Rot. von b gdw.
    |a|=|b| und a Substring vom zirkulären b



- Also:

```
CircRot(a,b): 1. if |a|!=|b| then return false;
              2. else return CircSearch(a, b)
```

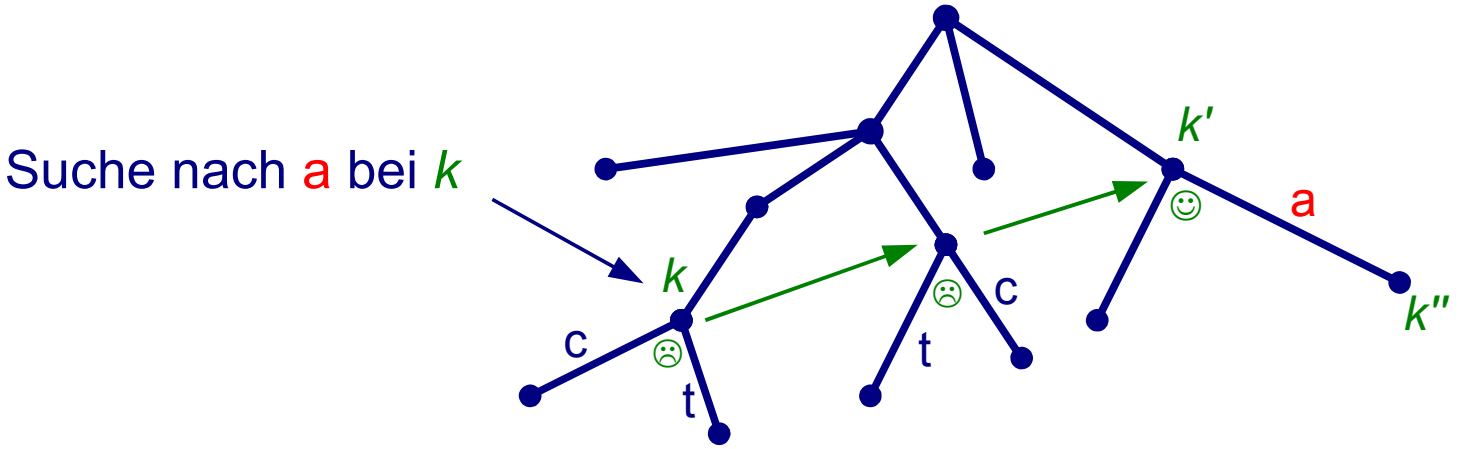
- Offenbar auch O(|a|+|b|)

Aufgabe 3 - Wettbewerb

- Algorithmus: Aho-Corasick, d.h.:
  - Keyword-Tree bauen
  - Nach allen Patterns auf einmal suchen
- “Am Ende programmieren alle C in Java”:
  - Baum und Template in je einem Array halten
  - Template-Datei nicht vorher parsen
    (sondern das ist gratis während der Suche)
  - Keine Klassen, keine Library-Funktionen, alles per Hand
    (ging eh schneller als API lesen)
  - keine Methodenrufe in innerer Schleife
- Algorithmus erweitert sodass nie Mismatches auftreten

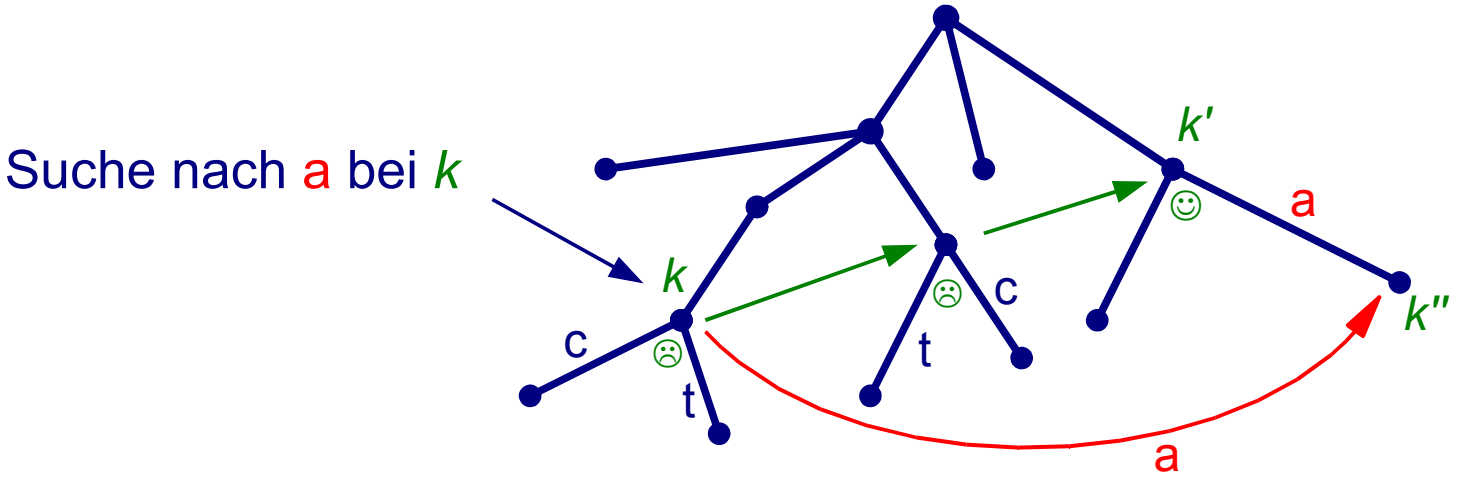
Erweiterung des Aho-Corasick Algorithmus

- Mismatch von x an Knoten k:
  - Failure Links ablaufen,
  - bis endlich ein Knoten k' das x an einer Kante (k', k'') hat
- Das ist ja immer der selbe, also vorher klar!



Erweiterung des Aho-Corasick Algorithmus

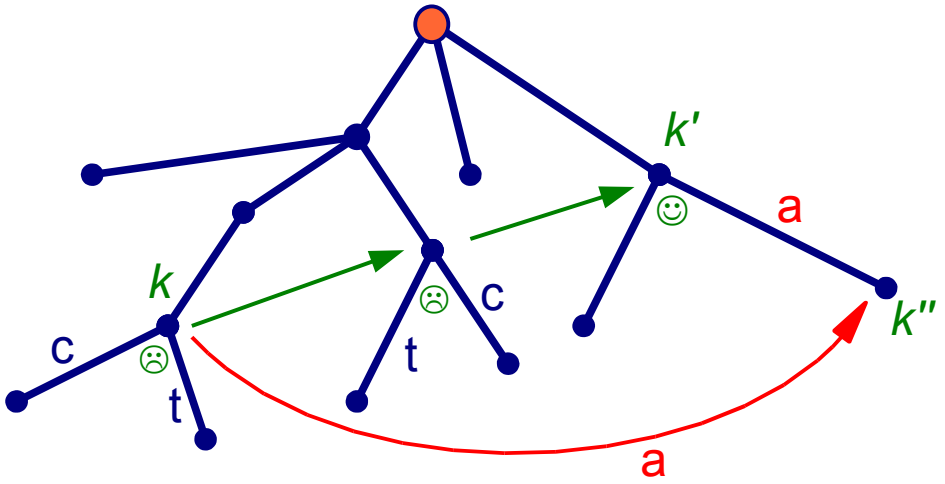
- Mismatch von x an Knoten k:
  - Failure Links ablaufen,
  - bis endlich ein Knoten k' das x an einer Kante (k', k'') hat
- Das ist ja immer der selbe, also vorher klar!



- Also: Neue Kante von k zu k''

Erweiterung des Aho-Corasick Algorithmus

- Folgen:
  - Jeder Knoten hat alle möglichen Kanten
  - Kanten können auch "nach oben" gehen, d.h. kein Baum mehr



  - Root zeigt mit den fehlenden Kanten auf sich selbst (das repräsentiert gerade die "finalen Mismatches")
  - ⇒ Bei der Suche: matches = |T|, mismatches = 0 (⇒FL obsolet!)
- Konstruktion der fehlenden Kanten:
  - Per Breitensuche als neue Phase nach FL-Konstruktion
  - Immer nur 1 FL pro Knoten folgen ⇒ O(n)

Algorithmus in Pseudocode

Kompletter Algorithmus

```
j := 1;                  // Next comparison in T
k := root(K);            // Root node of keyword tree
repeat
        while exists edge (k,k') with label(k,k')=T(j)
                if mark(k') ≠ NULL then
                        report mark(k');
                end if;
                z = out(k');
                while (z ≠ NULL)                // Check output links
                        report mark(z);         // Found a match
                        z = out(z);             // Recursion
                end if;
                k := k';                        // Down the
                j := j+1;                       // Check ne
        end while;
        if k=root(K) then       // Mismatch: move on in T
                j := j+1;
        else
                k := fl(k);     // Follow the failure link
        end if;
end repeat;
```

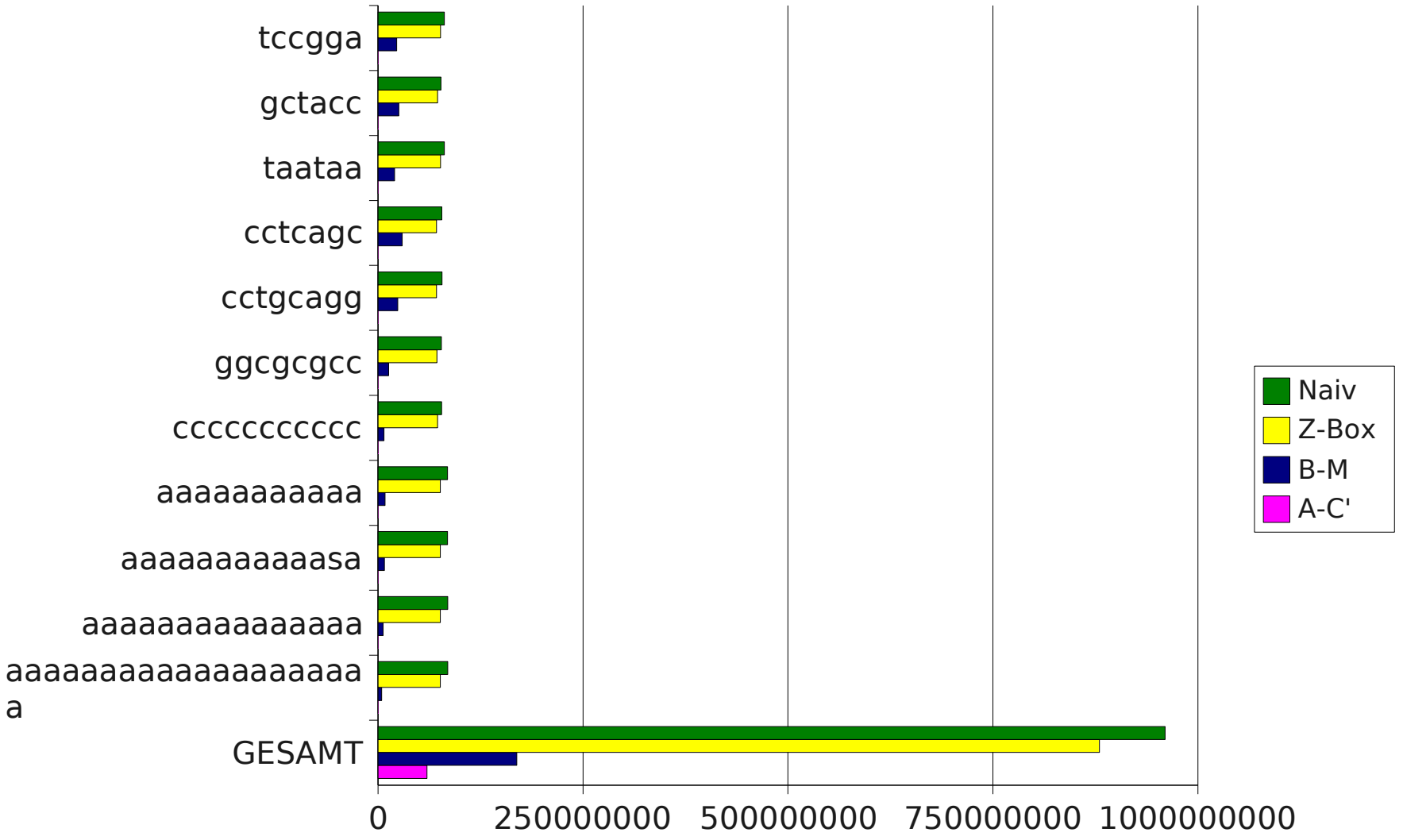
Pro Schleifendurchlauf:
1 Vergleich an Baumkante
Match: Weiter im Baum
Mismatch: FL folgen

```
j := 1;
k := root(K);

while j < eof
      k := child k' of k s.th. label(k,k')=T[j];
      if mark(k') ≠ NULL then
                report mark(k');
      end if;
      z = out(k);
      while (z ≠ NULL)
                report mark(z);
                z = out(z);
      end while;
      j++;
end while;
```

Pro Schleifendurchlauf:
1 Match an Baum-/Graph-Kante

Zeichenvergleiche



Laufzeiten in ms auf Amsel