CS681: Advanced Topics in Computational Biology

Can Alkan
EA509
calkan@cs.bilkent.edu.tr

http://www.cs.bilkent.edu.tr/~calkan/teaching/cs681/

Compression

- 1 – Reference based
  - Coding/decoding rather than real compression
  - Very high compression rate
  - Fast to encode
  - Slow to decode
  - Needs a reference genome
    - None, or poor quality for most species
    - Use same version of reference genome in decompression
  - Needs mapping (takes a long time)
    - Unmapped reads should be treated separately
    - Reads are mapped for other analyses anyway
  - CRAMtools/SAMtools, SlimGene, DeeZ, etc.
    - *Lossy*

CRAMtools / SAMtools

Post mapping; SAM format:

Read name *Flag* *Map* *Map quality* *CIGAR*

FCB01H4ABXX:6:2103:15210:113744 137 chr1 10001 0 90M = 10001 0
TAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCC
TAACCCTAACCCCAACCCCAACCCCAACCC

Read sequence

HHHHHGEEEGHHHGGBFGGGHGHHBEE?GECHHFHG9FFGF<DBFGGG<GGGGGAFGG
GGAEDFEDADA#####################

Read quality

X0:i:350 MD:Z:72T5T5T5 RG:Z:1 XG:i:0 AM:i:0 NM:i:3 SM:i:0 XM:i:3 XO:i:0
XT:A:R

edits

- Read name is unnecessary
- Flag tells you whether /1 or /2
- Map location and edit fields (CIGAR & MD) can be used to regenerate reads
- Don't store quality if edit distance = 0; otherwise only keep the qualities of changed bases

Fritz et al. Genome Research, 2011

CRAMtools / SAMtools

Post mapping; SAM format:

Read name *Flag* *Map* *Map quality* *CIGAR*

FCB01H4ABXX:6:2103:15210:113744 137 chr1 10001 0 90M = 10001 0
TAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCC
TAACCCTAACCCCAACCCCAACCCCAACCC *Read sequence*

HHHHHGEEEGHHHGGBFGGGHGHHBEE?GECHHFHG9FFGF<DBFGGG<GGGGGAFGG
GGAEDFEDADA####################### *Read quality*

X0:i:350 MD:Z:72T5T5T5 RG:Z:1 XG:i:0 AM:i:0 NM:i:3 SM:i:0 XM:i:3 XO:i:0
XT:A:R

Keep: **137** ; **chr1:10001** ; **0** ; **90M**; **72T5T5T5** ; **(#,#,#)**
Add a layer of Huffman encoding

Fritz et al. Genome Research, 2011

CRAMtools: test case

- One human genome
  - 40X coverage
  - 134 GB gzipped = 479 GB raw text
  - Mapped with BWA; >1 day with 30 CPUs
  - SAM format converted to BAM file: 112 GB
  - BAM to CRAM: 7.5 GB
  - Decode CRAM to BAM: 33 GB (lossy!!!)

Compression

- 2 – Reference free
  - Less compression rate
  - No need for reference, applicable to any dataset from any species
  - Slower to compress, faster to decompress
  - Can be lossy or lossless
  - Multipurpose compressors:
    - gzip, bzip2, 7-zip, etc.
  - Specialized FASTQ compressors
    - SCALCE, ReCoil, G-SQZ, etc.

Reference-free compression

- Easy task (or gzip, etc.): Concatenate all sequences, then run Lempel-Ziv algorithm
- Problem: Locality

Lempel-Ziv Compression

a b b a a b b a a b a b b a a a a b a a b b a
0 1 1 0 2--- 4--- 2--- 6------- 5--- 5--- 7------- 3--- 0

Index	Entry	Index	Entry
0	a	7	baa
1	b	8	aba
2	ab	9	abba
3	bb	10	aaa
4	ba	11	aab
5	aa	12	baab
6	abb	13	bba

Reordering improves locality

File Size: 250MB, 5Mil 51bp Bacterial Genome

Pre-processing	Time (s)	Gzip time	Size (MB)	Comp. Factor	Boosting
-	-	70	65	4	-
Mapping	180	21	20	12.5	3.25
Lexo. Sorting	10	30	26	9.61	2.5
Cores*	10	21	21	11.9	3.1

*** Idea behind SCALCE**

Reordering example

Ref: AAAAAATGACGTCTCTCCTCCTTTTTTAAAACCT

Original	Mapping	Sorting	Cores
CTTTTT	AAAAAA	AAAAAA	AAAAAA
GATGAC	TAATGA	ATGACG	TAAAAC
CCCCCT	GATGAC	CCCCCT	CCCCCT
AAAAAA	ATGACG	CTTTTT	CTTTTT
ATGACG	CCCCCT	GATGAC	TAATGA
TAAAAC	CTTTTT	TAAAAC	GATGAC
TAATGA	TAAAAC	TAATGA	ATGACG

Cores: Locally Consistent Parsing

LCP (Sahinalp STOC 1994, Sahinalp FOCS 1996) is a combinatorial pattern matching technique that aims to identify building blocks of strings. For any user-specified integer *c* and with any alphabet, the LCP identifies core substrings of length between *c* and *2c* such that:

- any string from the alphabet of length *3c* or more include at least one such core string
- there are no more than three such core strings in any string of length *4c* or less
- if two long substrings of a string are identical, then their core substrings must be identical

Increasing Locality

- Goal: Obtain a few core substrings for each read so that two highly overlapping reads will have common core substrings. We obtain a set of core strings such that
  - A long prefix of a core substring can not be a suffix of another core substring (this assures that two subsequent core substrings can not be too close to each other).
  - Each read includes at least one core substring.

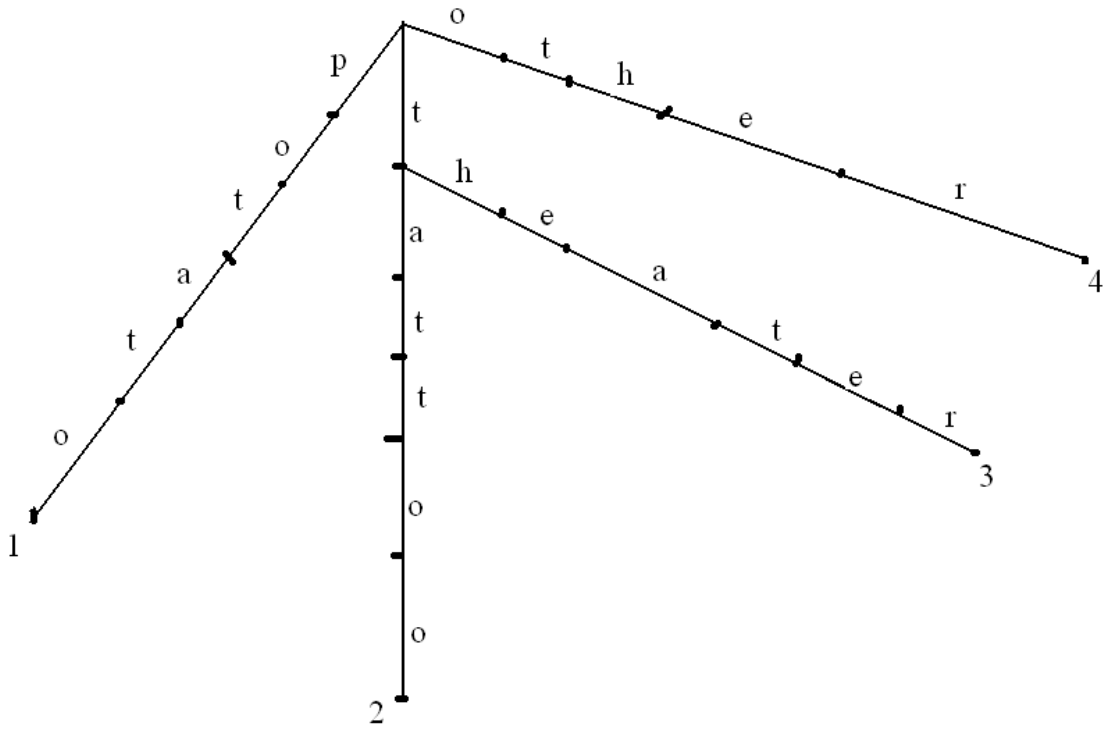
Finding cores

Find all “core substrings" in a given read and place it in a bucket which has the maximum number of reads.

- Trie data structure: finding all core substrings within a read would require O(cr) time (r: read length, c: length of all core substrings in that read).
- Improvement: Aho-Corasick dictionary matching algorithm using an automaton. O(r+k), where k is the number of core substring occurrences in each read.
- More improvement: Alphabet is small, and number of core substrings is fixed; pre-process automaton to calculate bucket in O(1) time, reduce total search time to O(r).

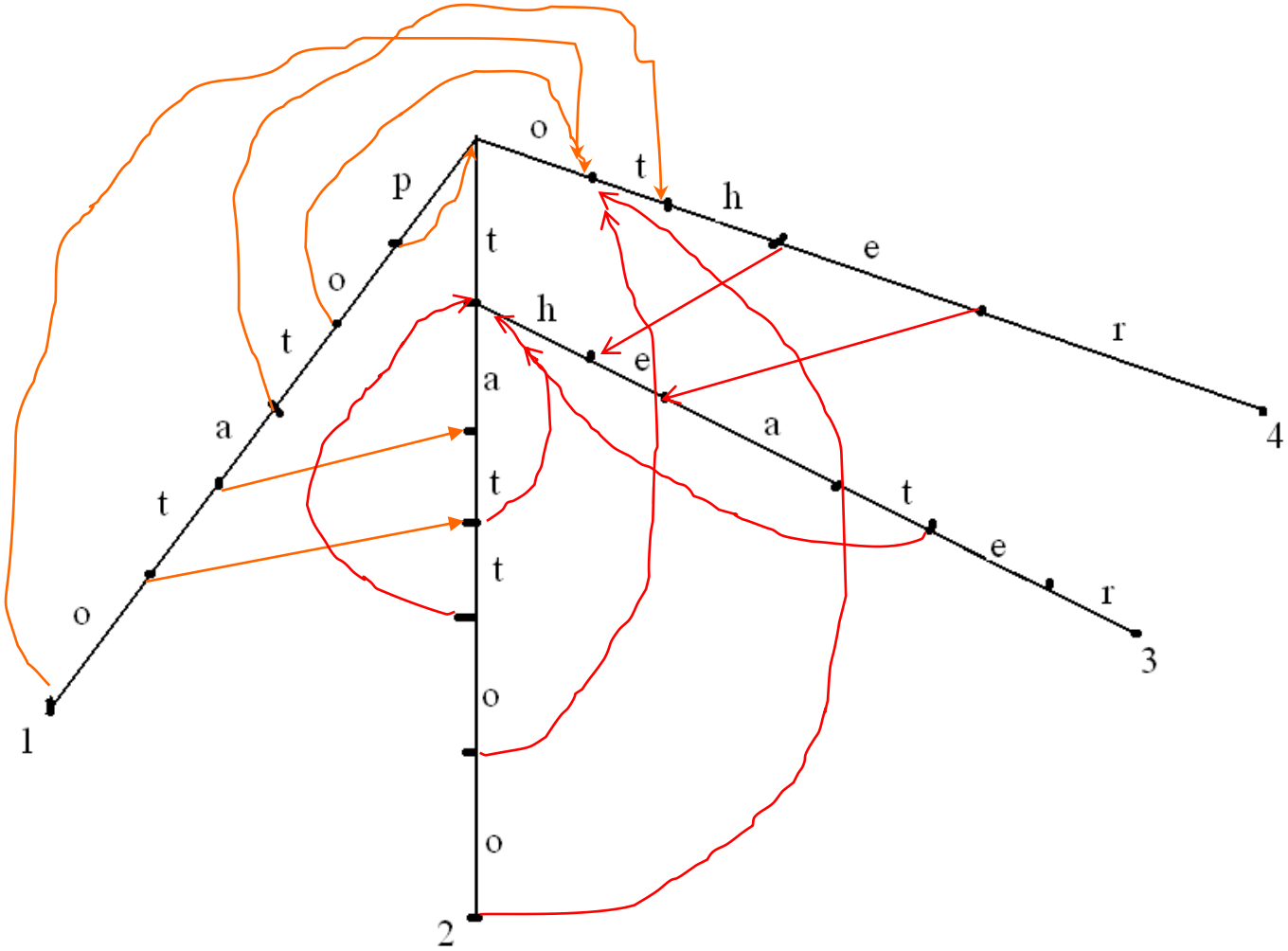
Trie data structure

P={potato, tattoo, theater, other}



Failure links

P={potato, tattoo, theater, other}

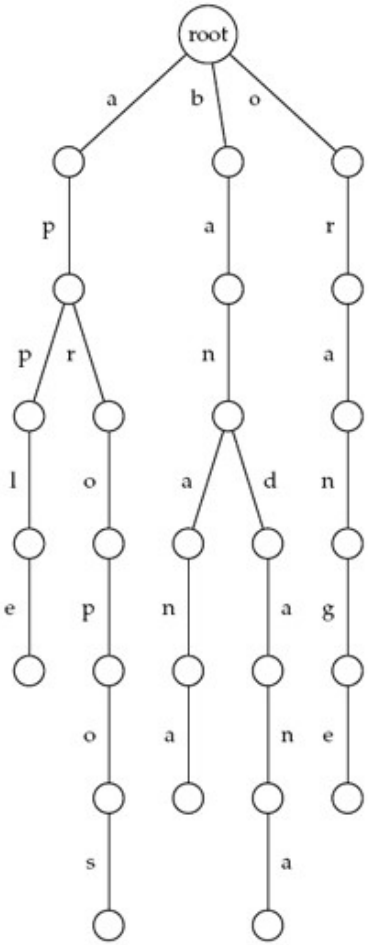


Slides from Charles Yan

AHO-CORASICK

Search in keyword trees

- Naïve threading in keyword trees do not *remember* the partial matches
- P={apple, appropos}
- T=appappropos
- When threading
  - *app* is a partial match
  - But naïve threading will go back to the root and re-thread *app*
- Define *failure links*



Failure Link

v: a node in keyword tree K

L(v): the label on v, that is, the concatenation of characters on the path from the root to v.

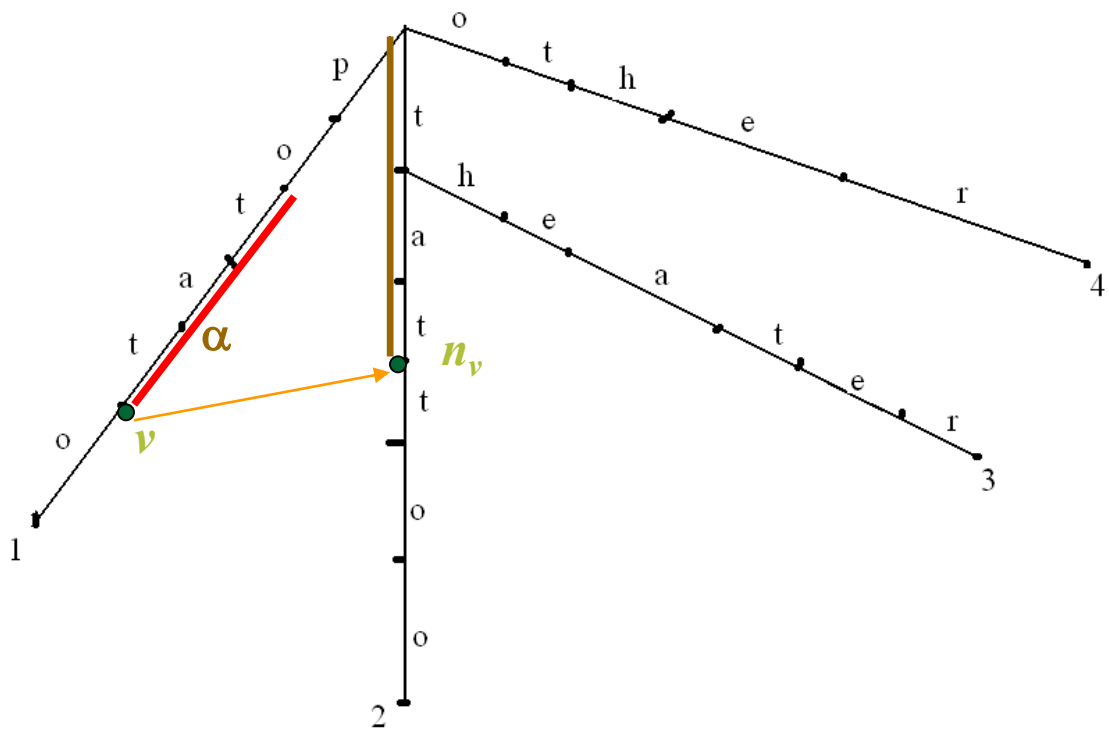
lp(v): the length of the longest proper suffix of string L(v) that is a prefix of some pattern in P. Let this substring be α.

Lemma. There is a unique node in the keyword tree that is labeled by string α. Let this node be n_v. Note that n_v can be the root.

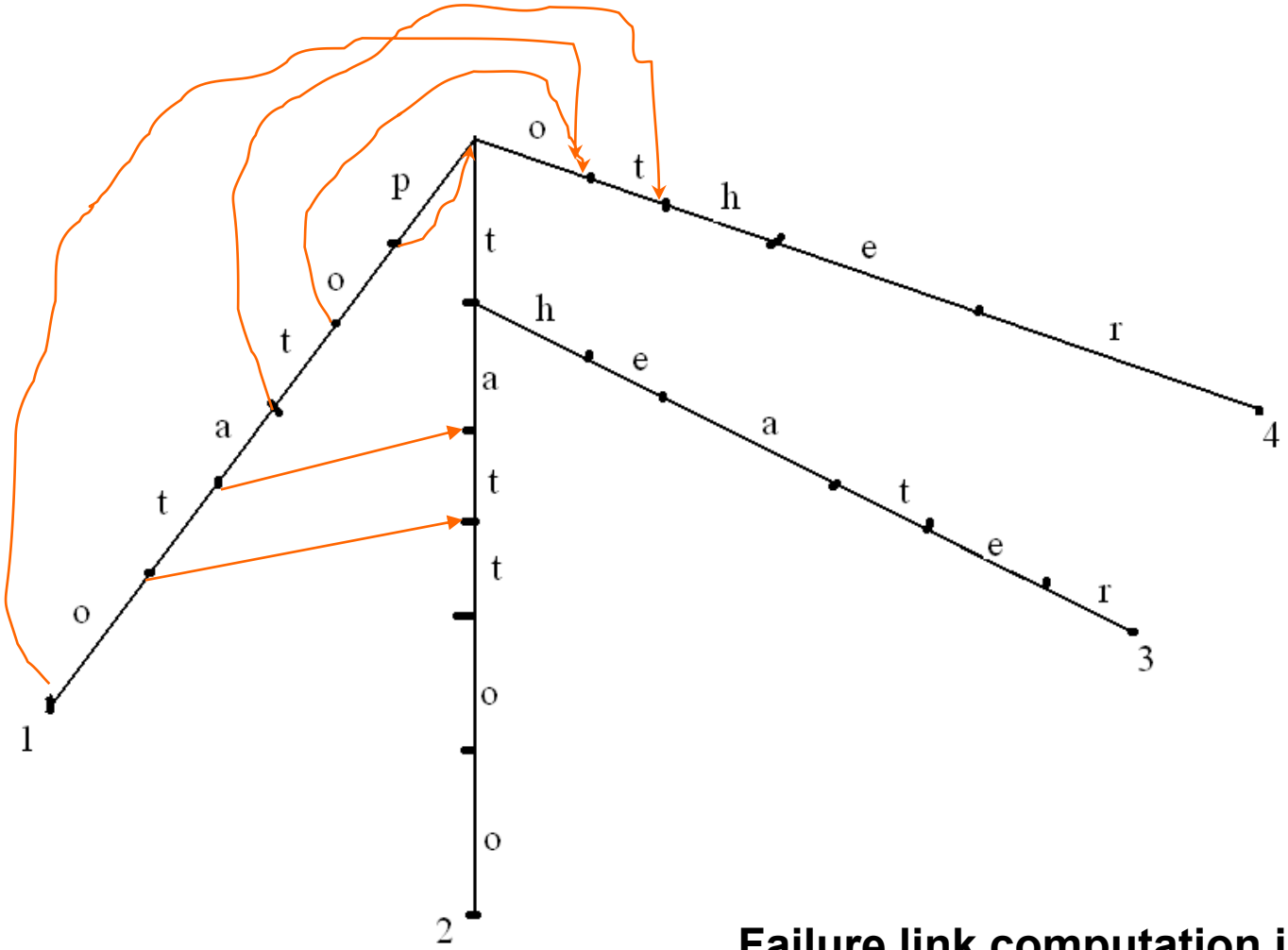
The ordered pair (v, n_v) is called a **failure link**.

Failure Link

P={potato, tattoo, theater, other}

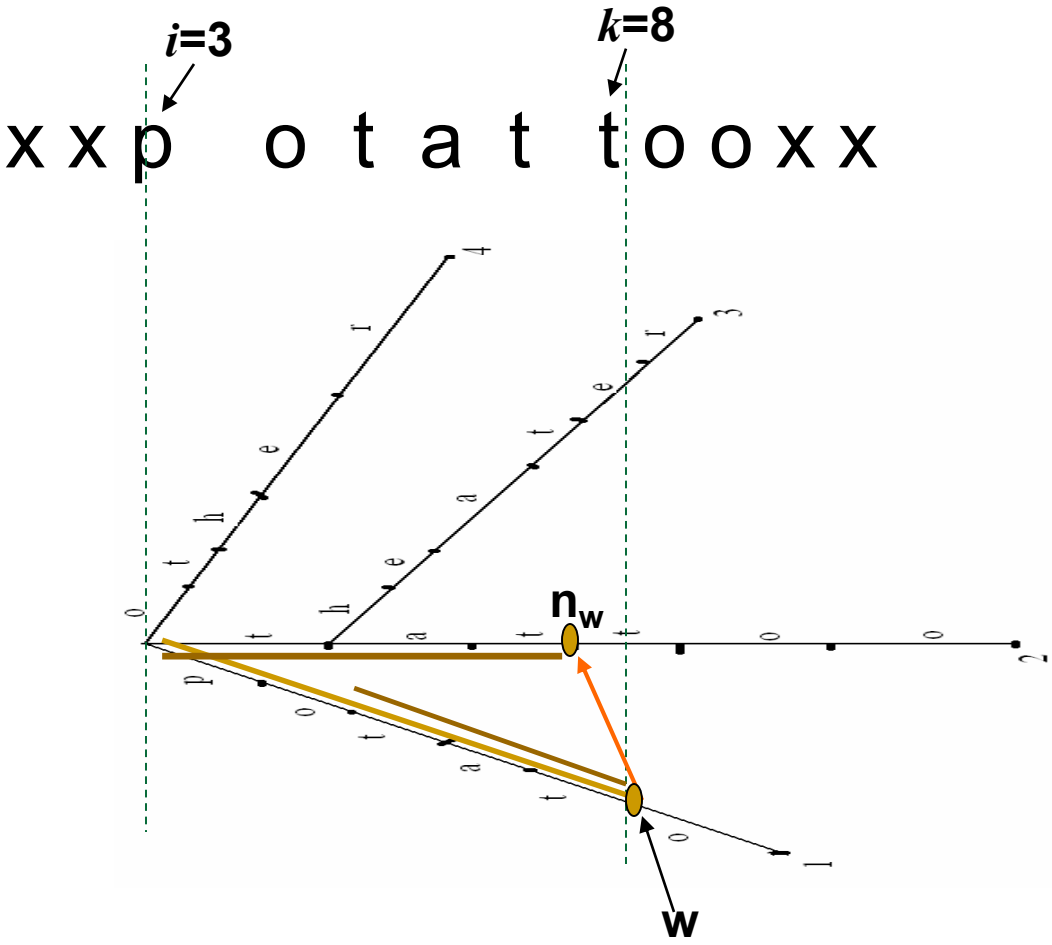


Failure Link

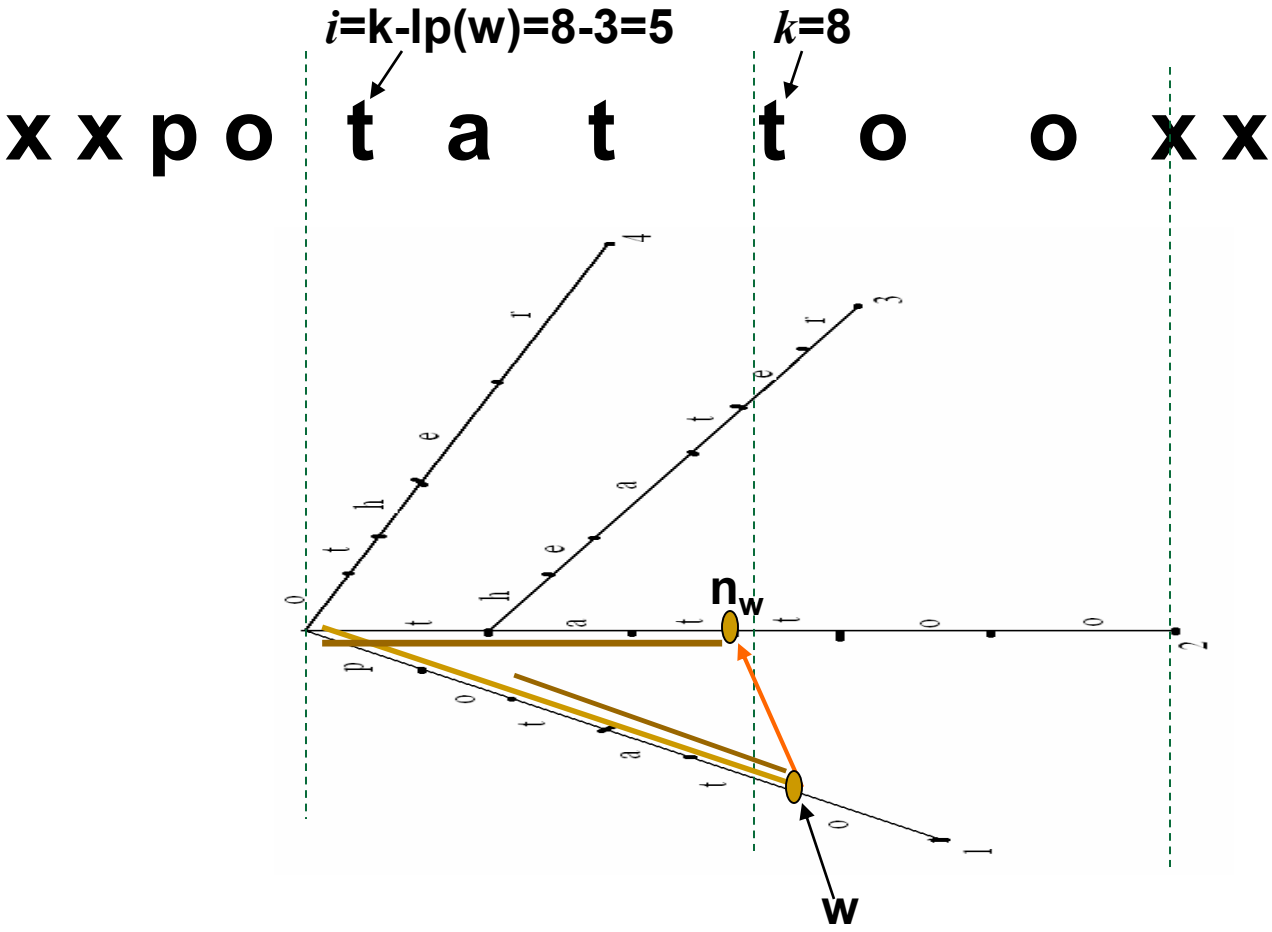


Failure link computation is O(n)

Failure Link



Failure Link



Failure Link

How to construct failure links for a keyword tree in a linear time?

Let d be the distance of a node (v) from the root r.

When $d \leq 1$, i.e., v is the root or v is one character away from r, then $n_v = r$.

Suppose n_v has been computed for every node (v) with $d \leq k$, we are going to compute n_v for every node with $d = k+1$.

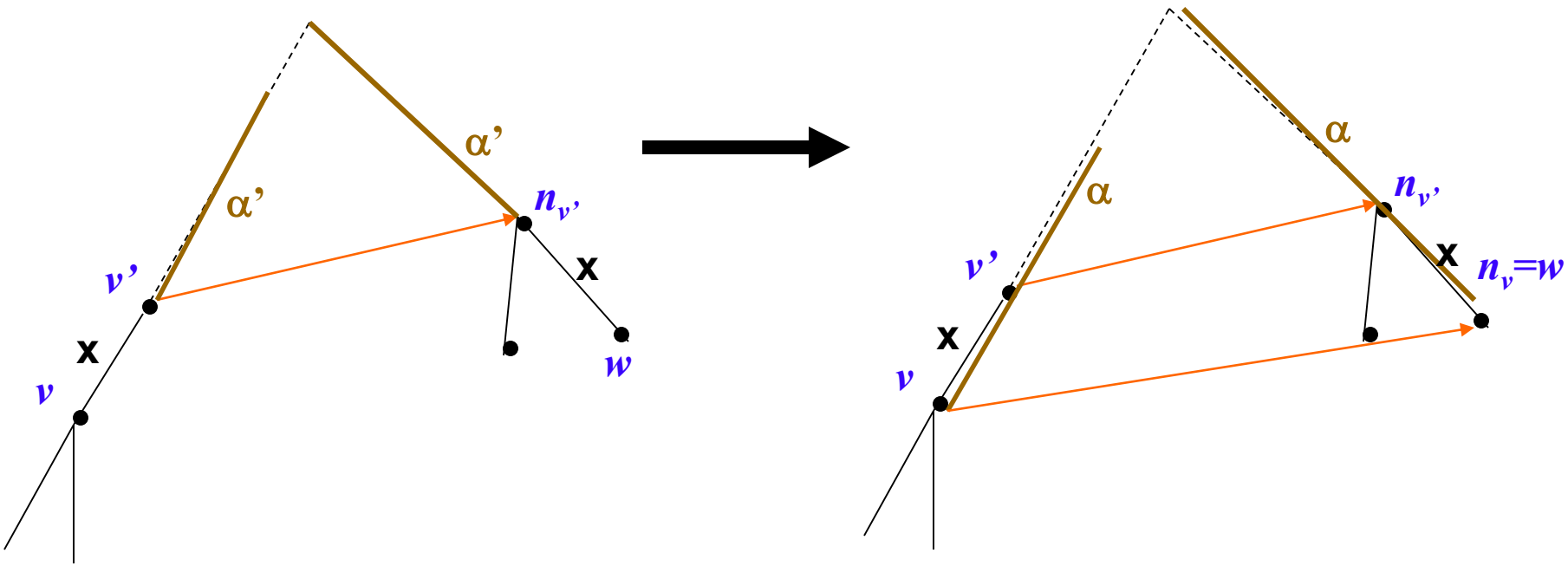
v': parent of v, then v' is k characters from r, that is d=k

thus the failure link for v' ($n_{v'}$) has been computed.

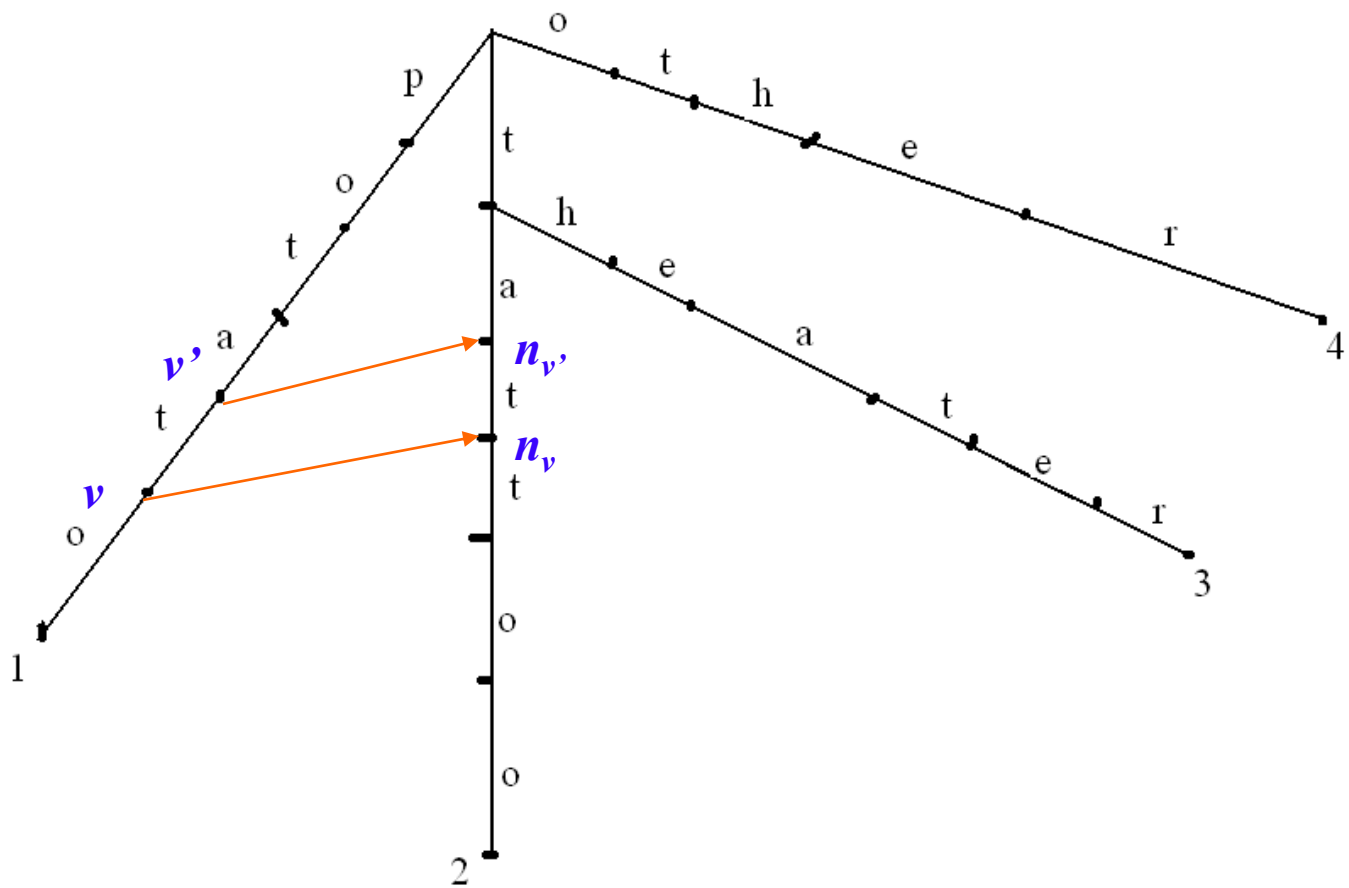
x: the character on edge (v', v)

Failure Link

(1) If there is an edge ($n_{v'}$, w) out of $n_{v'}$ labeled with x, then n_v=w.

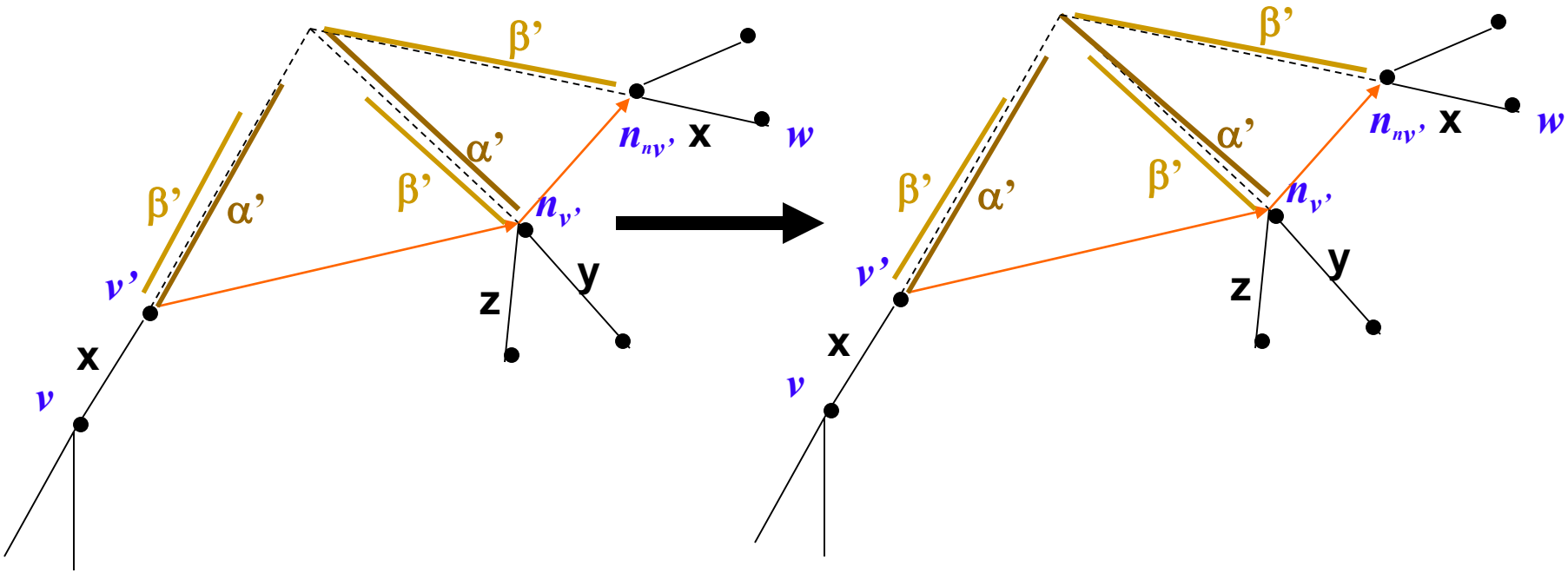


Failure Link



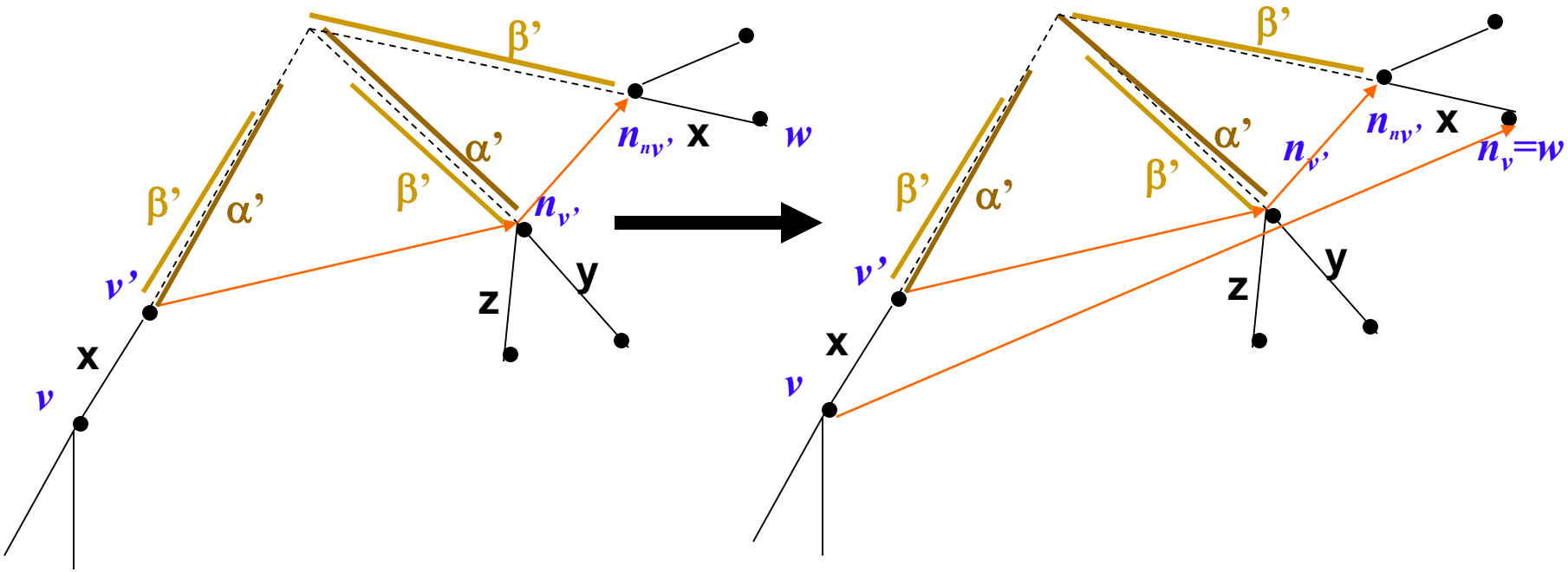
Failure Link

(2) If such an edge does not exist, examine $n_{nv'}$ to see if there is an edge out of it labeled with x. Continue until the root.

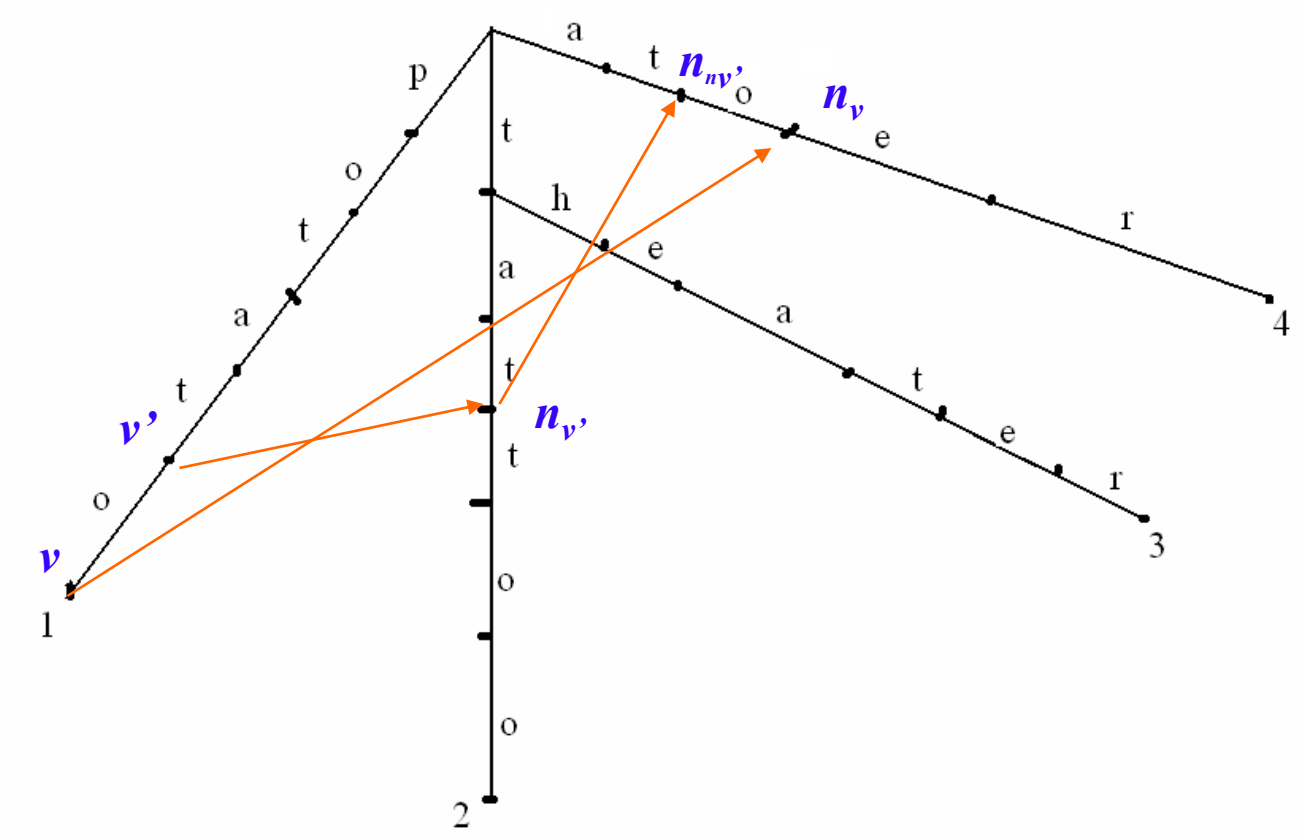


Failure Link

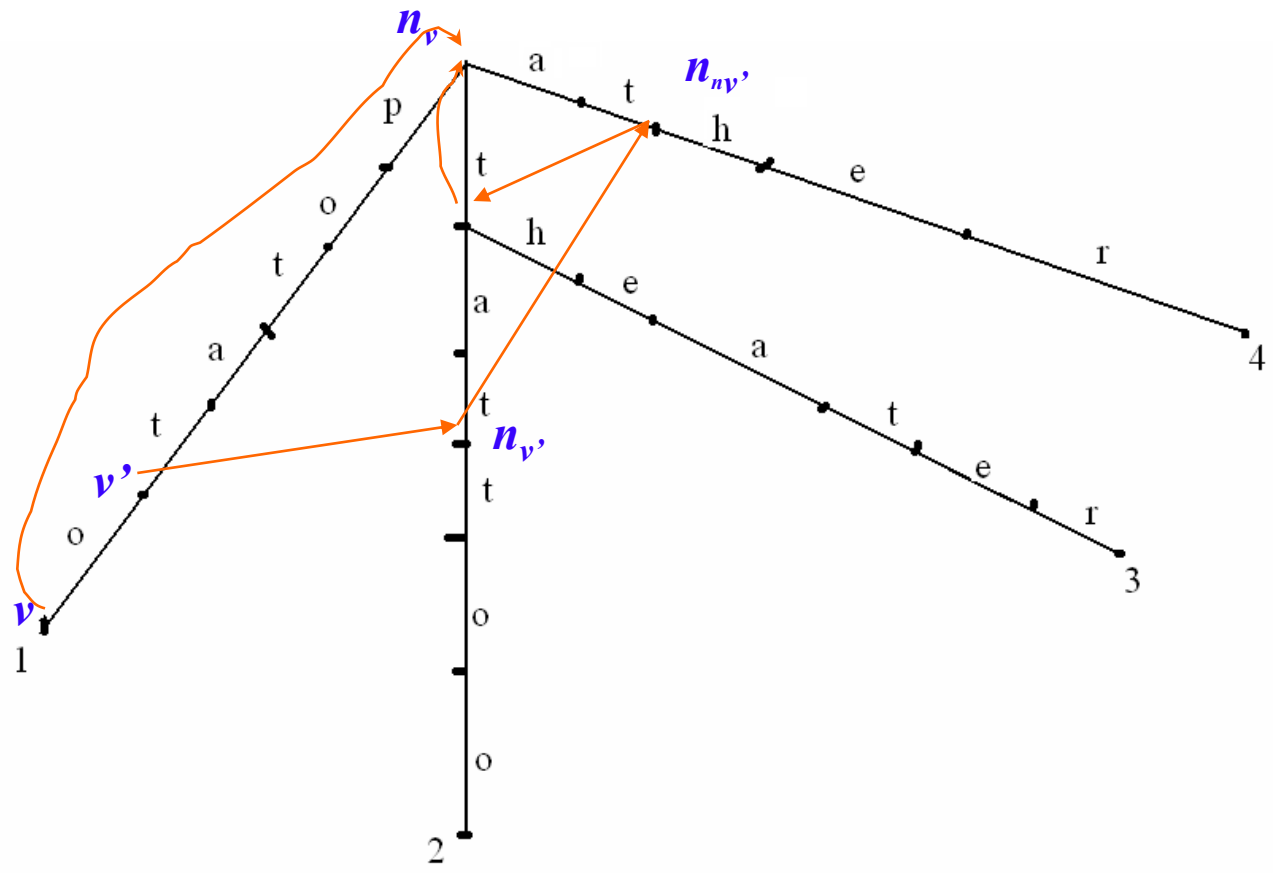
(2) If such an edge does not exist, examine $n_{nv'}$ to see if there is an edge out of it labeled with x. Continue until the root.



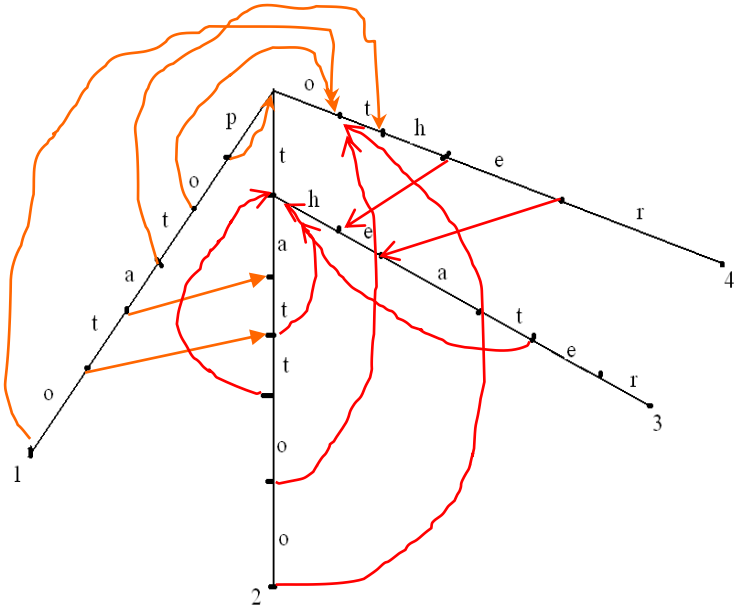
Failure Link



Failure Link



Aho-Corasick Algorithm

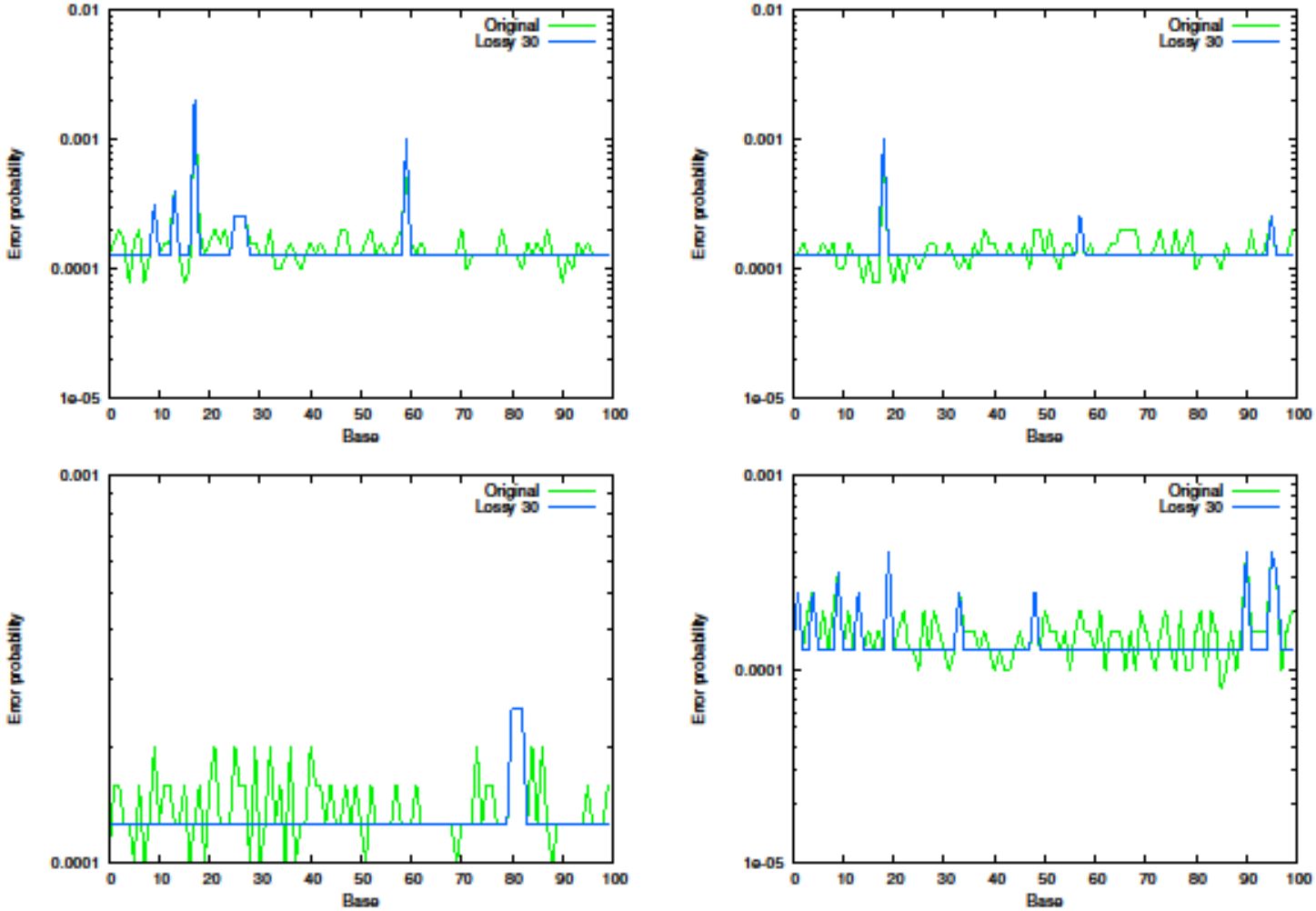


Input: Pattern set P and text T
Output: all occurrences in T any pattern from P
Algorithm AC
l=1;
c=1;
w=root
Repeat
    while there is an edge (w, w') labeled with T(c)
        if w' is numbered by pattern i then
            report that p_i occurs in T starting at l;
        w=w'; c++;
    w=n_w and l=c-lp(w);
Until c>m

Quality Score Transformation

- Sequence alphabet has 5 characters (A,C,G,T,N); but quality string alphabet is larger, thus compresses less
  - Generate qualities with a smaller alphabet to improve compression
- Expect some small noise in a normal run of sequencing machine.
- Calculate the frequency of the alphabet and reduce the noise by merging the local maxima up to *e%* threshold.

(*optional*) Quality Score Transformation



Original and transformed quality scores for four random reads that are chosen from NA18507 individual.

Test case

Dataset			gzip		SCALCE (lossless)			SCALCE (lossy 30%)		
Name	Number of reads	Size	Size	Rate	Size	Rate	Boosting factor	Size	Rate	Boosting factor
P.aeruginosa RNAseq	89M	10 076	3183	3.17	1496	6.74	2.13×	953	10.58	3.34×
P.aeruginosa genomic	81M	9163	3211	2.85	1655	5.54	1.94×	1126	8.14	2.85×
NA18507 WGS	1.4B	300 337	113 132	2.65	76 890	3.91	1.47×	58 031	5.18	1.95×
NA18507 single lane	36M	7708	3058	2.52	2146	3.59	1.42×	1639	4.70	1.86×

Name	BEETL time (min)	BEETL size	SCALCE time (min)	SCALCE size
P.aeruginosa RNAseq	29	197	8	95
P.aeruginosa Genomic	31	257	6	137
NA18507 single lane	51	448	10	412

For more

Comparison of high-throughput sequencing data compression tools

Ibrahim Numanagić, James K Bonfield, Faraz Hach, Jan Voges, Jörn Ostermann, Claudio Alberti, Marco Mattavelli & S Cenk Sahinalp

Nature Methods **13**, 1005–1008 (2016) | Download Citation