Disk compression of *k*-mer sets

Workshop on Compression, Text and Algorithms
(WCTA 2021)

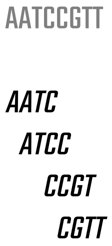
Amatur Rahman[1], Rayan Chikhi[2], Paul Medvedev[1,3,4]

[1]Department of Computer Science and Engineering, Penn State
[2]Department of Computational Biology, C3BI USR 3756 CNRS, Institut Pasteur, Paris, France
[3]Department of Biochemistry and Molecular Biology, Penn State
[4]Center for Computational Biology and Bioinformatics, Penn State

Compression of a set of *k*-mers

Methods based on *k*-mers are everywhere

- metagenomics (e.g. Kraken)
- genome assembly (e.g. Spades)
- sequence divergence (e.g. Mash)
- genotyping (e.g VarGeno, MALVA)
- database search (e.g. SBTs)
- variant calling (e.g. Kevlar)



AATCCGTT

AATC

ATCC

CCGT

CGTT

Compression of a set of *k*-mers

Methods based on *k*-mers are everywhere

- metagenomics (e.g. Kraken)
- genome assembly (e.g. Spades)
- sequence divergence (e.g. Mash)
- genotyping (e.g VarGeno, MALVA)
- database search (e.g. SBTs)
- variant calling (e.g. Kevlar)

AATCCGTT

AATC
ATCC
CCGT
CGTT

How large can a set of k-mer get?

12 TB

BIGSI
Database

31-mers for
450 million microbial genomes

Compression of a set of *k*-mers

Methods based on *k*-mers are everywhere

- metagenomics (e.g. Kraken)
- genome assembly (e.g. Spades)
- sequence divergence (e.g. Mash)
- genotyping (e.g VarGeno, MALVA)
- database search (e.g. SBTs)
- variant calling (e.g. Kevlar)

AATCCGTT

AATC
ATCC
CCGT
CGTT

How large can a set of k-mer get?

12 TB

BIGSI
Database

31-mers for
450 million microbial genomes

Non-negligible write
and load time

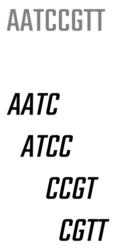
High storage cost

Slower transfer
across network

Compression of a set of *k*-mers

Methods based on *k*-mers are everywhere

- metagenomics (e.g. Kraken)
- genome assembly (e.g. Spades)
- sequence divergence (e.g. Mash)
- genotyping (e.g VarGeno, MALVA)
- database search (e.g. SBTs)
- variant calling (e.g. Kevlar)


AATCCGTT

AATC
ATCC
CCGT
CGTT

How large can a set of k-mer get?

12 TB

BIGSI
Database

31-mers for
450 million microbial genomes

Non-negligible write
and load time

High storage cost

Slower transfer
across network

Solution? **Disk compression.**

Compression of a set of *k*-mers

Methods based on *k*-mers are everywhere

- metagenomics (e.g. Kraken)
- genome assembly (e.g. Spades)
- sequence divergence (e.g. Mash)
- genotyping (e.g VarGeno, MALVA)
- database search (e.g. SBTs)
- variant calling (e.g. Kevlar)

```
AATCCGTT

AATC
 ATCC
   CCGT
    CGTT
```

How large can a set of k-mer get?

12 TB

BIGSI
Database

31-mers for
450 million microbial genomes

Non-negligible write and load time

High storage cost

Slower transfer across network

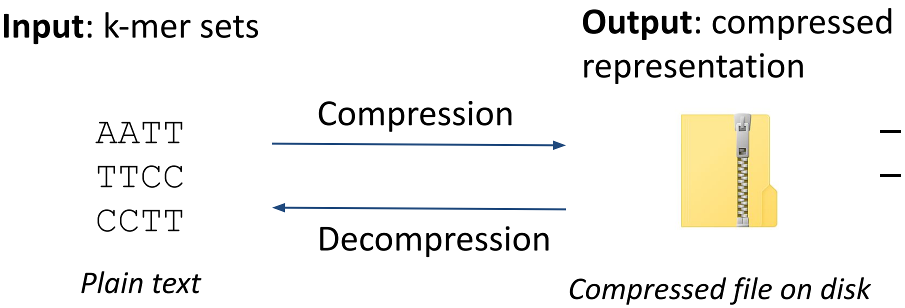
Solution? **Disk compression.**

Why not just use a membership data-structure?

- Designed to support direct query
- Dropping this requirement saves space...

★ BOSS and variants
★ BFT
★ Fully dynamic dBGs
★ UST-Compress

Disk compression of k-mer sets



Input: k-mer sets

Output: compressed representation

```
AATT
TTCC
CCTT
```

Plain text

Compression

Decompression

Compressed file on disk

- less space on disk
- supports fast decompression
  - But no direct query

Why not just use popular compressors?

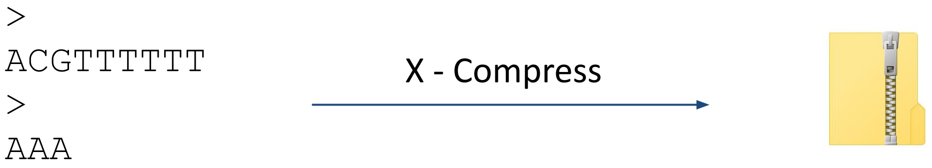
- General purpose compressor
  - gzip, bzip2, lzma
- Special compressor for sequences: already being used to compress reads
  - MFCompress, DELIMINATE, NAF

These techniques do not exploit inherent redundancy in k-mer sets fully

Why use “**k-mer**” compression? Can’t we just compress reads?

- Reads/sequence compression tools: MFC, DELIMINATE, NAF etc. → X

Compression Process



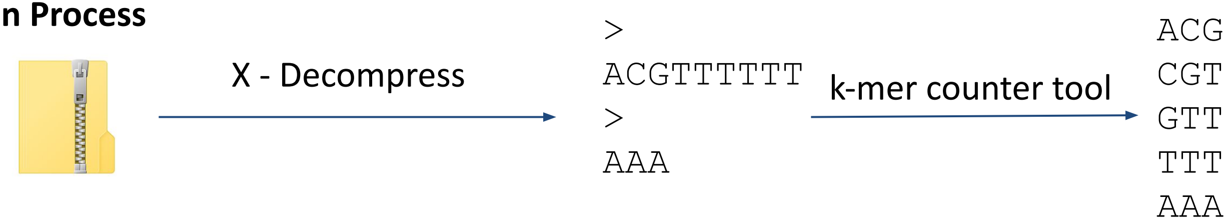
```
>
ACGTTTTTT
>
AAA
```

X - Compress

Reads in FASTA-format

Compressed file on disk

Decompression Process



X - Decompress

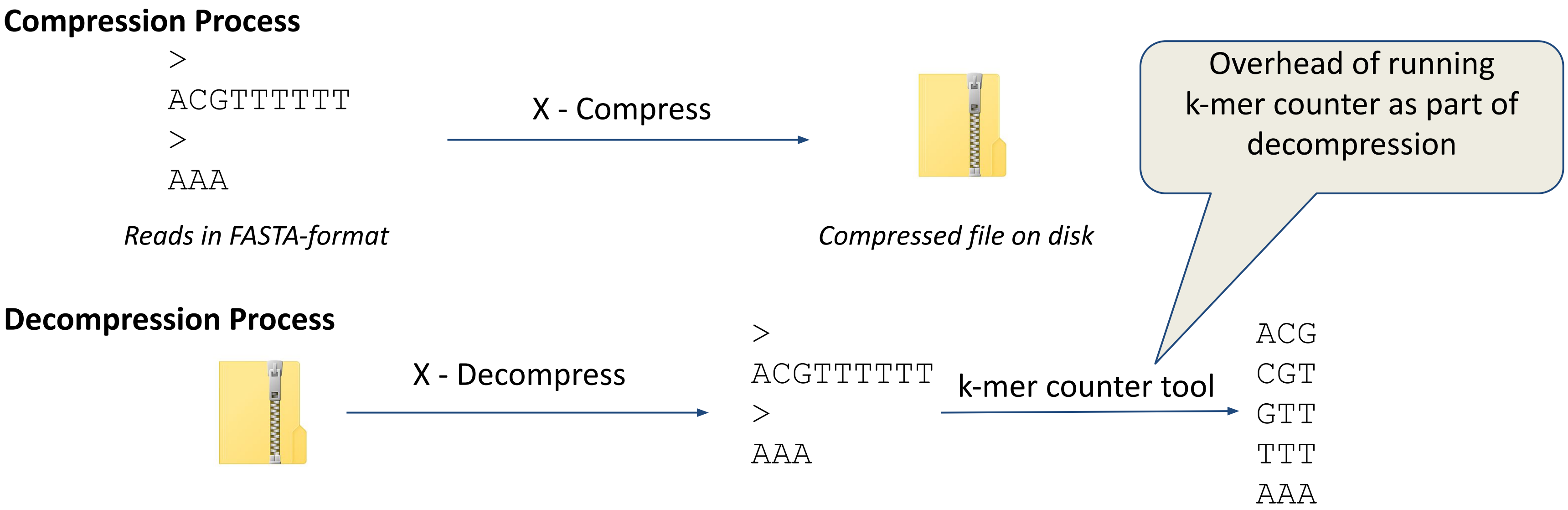
```
>
ACGTTTTTT
>
AAA
```

k-mer counter tool

```
ACG
CGT
GTT
TTT
AAA
```

Why use “**k-mer**” compression? Can’t we just compress reads?

- Reads/sequence compression tools: MFC, DELIMINATE, NAF etc. $\longrightarrow X$



Compression Process

```
>
ACGTTTTTT
>
AAA
```

Reads in FASTA-format

X - Compress

Compressed file on disk

Overhead of running k-mer counter as part of decompression

Decompression Process

X - Decompress

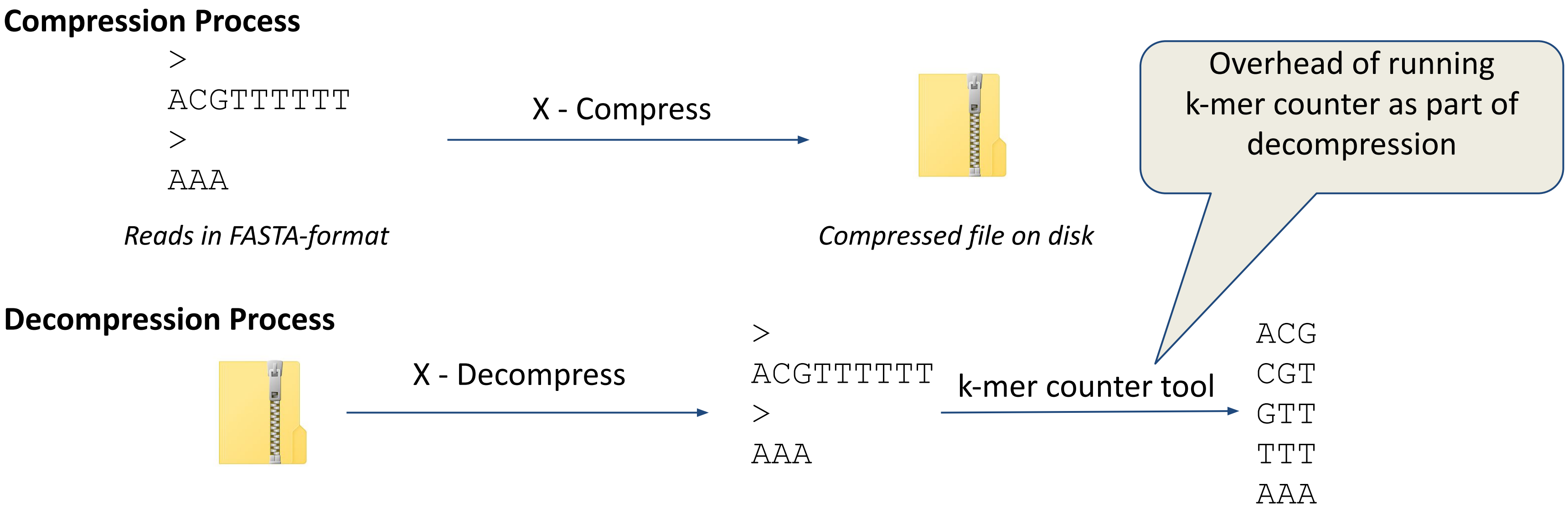
```
>
ACGTTTTTT
>
AAA
```

k-mer counter tool

```
ACG
CGT
GTT
TTT
AAA
```

Why use “**k-mer**” compression? Can’t we just compress reads?

- Reads/sequence compression tools: MFC, DELIMINATE, NAF etc. $\longrightarrow X$



Compression Process

```
>
ACGTTTTTT
>
AAA
```

Reads in FASTA-format

X - Compress

Compressed file on disk

Overhead of running k-mer counter as part of decompression

Decompression Process

X - Decompress

```
>
ACGTTTTTT
>
AAA
```

k-mer counter tool

```
ACG
CGT
GTT
TTT
AAA
```

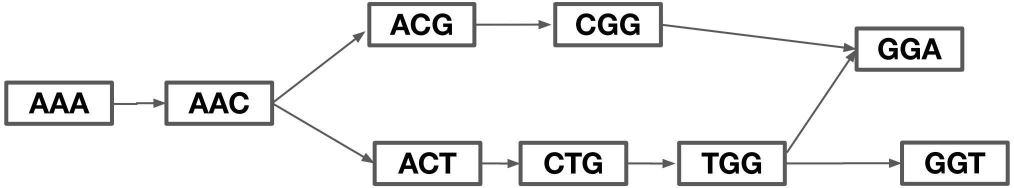
When k-mer set is not related to reads:

- universal hitting set (set of k-mers that hit a L-long sequence) *(Orenstein et al, 2017)*
- chromosome-specific reference dictionary *(Rangavittal et al, 2019)*
- winnowed min-hash sketch *(Sahlin et al, 2019)*

(node centric) de Bruijn graph

Given a set of *k*-mers S, dBG(S) is a directed graph where

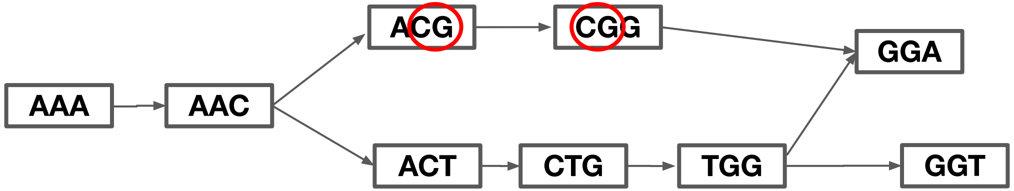
- Nodes are the *k*-mers
- Edge x→y iff
  - the suffix of length *k-1* of *x* is equal to the prefix of length *k-1* of *y*



(node centric) de Bruijn graph

Given a set of *k*-mers S, dBG(S) is a directed graph where

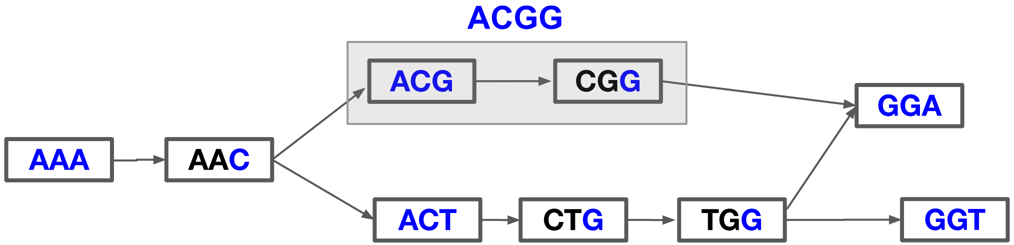
- Nodes are the *k*-mers
- Edge x→y iff
  - the suffix of length *k-1* of *x* is equal to the prefix of length *k-1* of *y*



(node centric) de Bruijn graph

Given a set of *k*-mers S, dBG(S) is a directed graph where

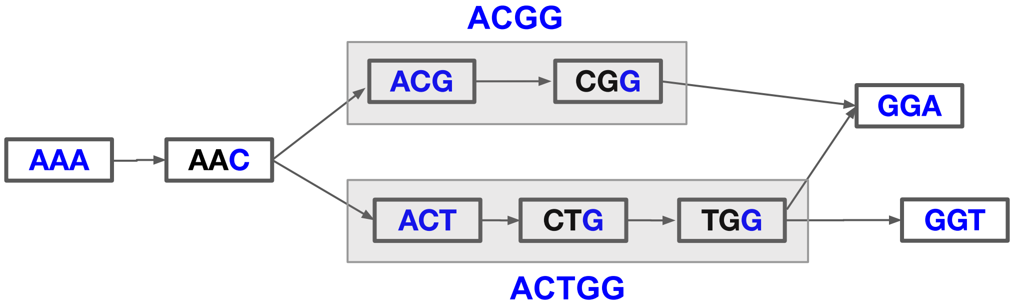
- Nodes are the *k*-mers
- Edge x→y iff
  - the suffix of length *k-1* of *x* is equal to the prefix of length *k-1* of *y*



(node centric) de Bruijn graph

Given a set of *k*-mers S, dBG(S) is a directed graph where

- Nodes are the *k*-mers
- Edge x→y iff
  - the suffix of length *k-1* of *x* is equal to the prefix of length *k-1* of *y*



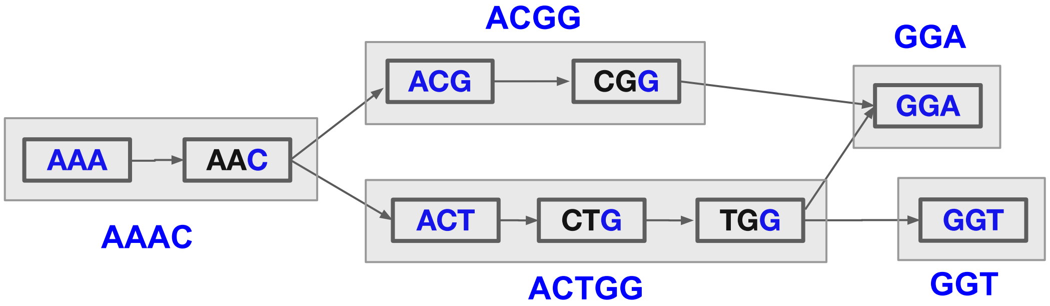
(node centric) de Bruijn graph

Given a set of *k*-mers S, dBG(S) is a directed graph where

- Nodes are the *k*-mers
- Edge x→y iff
  - the suffix of length *k-1* of *x* is equal to the prefix of length *k-1* of *y*

Unitigs:

- Non-branching paths in dBG (gray)



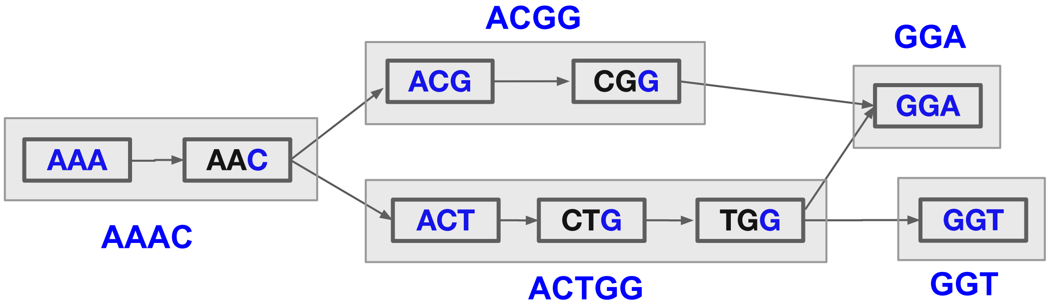
(node centric) de Bruijn graph

Given a set of *k*-mers S, dBG(S) is a directed graph where

- Nodes are the *k*-mers
- Edge x→y iff
  - the suffix of length *k-1* of *x* is equal to the prefix of length *k-1* of *y*

Unitigs:

- Non-branching paths in dBG (gray)
- Spelling of unitigs is a way to represent the k-mers in less space
  - Generalizes to Spectrum-Preserving String Sets *(Rahman and Medvedev, RECOMB 2020)*
    - contain the same k-mers as S and only contain them once

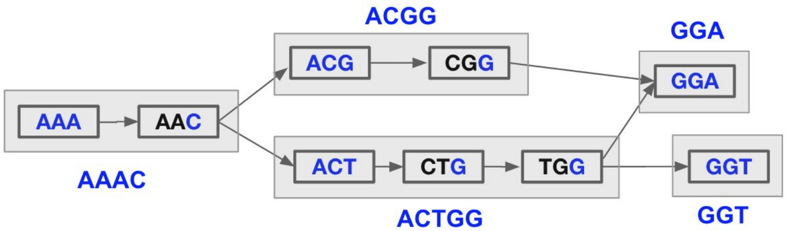


Spectrum-preserving string sets

A set of strings are called a ***spectrum-preserving string set (SPSS) representation*** if

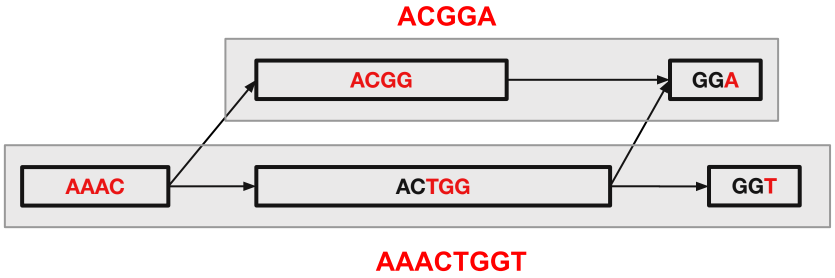
- They contain the same k-mers as S and only contain them once *(Rahman and Medvedev, RECOMB 2020, Brinda, Baym and Kucherov, 2020)*

Representation 1 (unitig based)



weight = 19

Representation 2 (SPSS by UST-Compress)

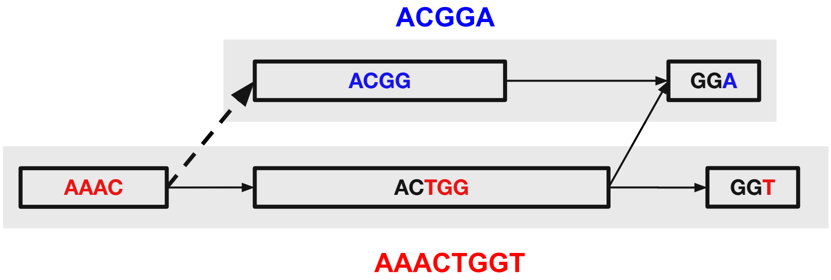


weight = 13

- In our previous work, we proposed a greedy algorithm (**UST-Compress**) to find low-weight SPSS.

- Now, we take an approach that builds on it.

From SPSS to ESS: Enriched String Set representation



SPSS:

- Only allows DNA characters (A,C,G,T)

ACGGA
AAACTGGT

ESS:

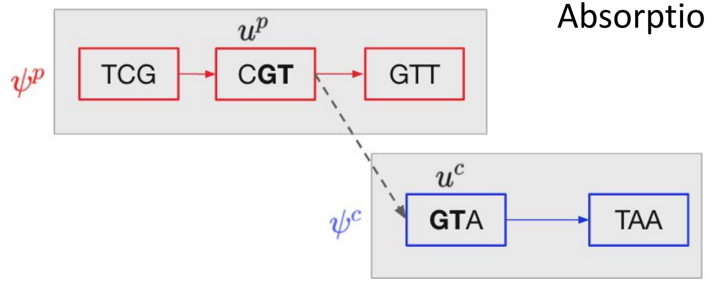
- 3 extra character [,], +

AAAC[ACGGA]TGGT

↓

AAAC[+GGA]TGGT

ESS-Compress representation

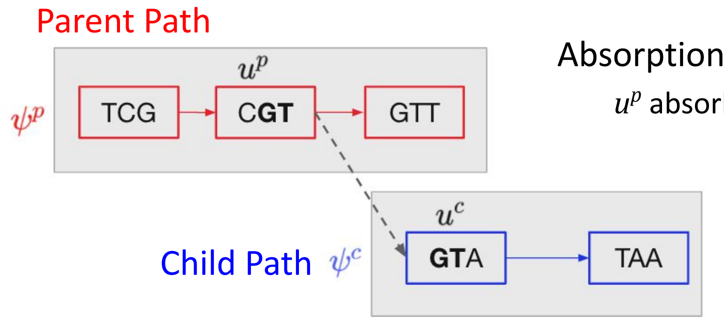


Absorption edge - - - - -

TCGTT
GTAA

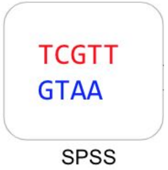
SPSS

ESS-Compress representation

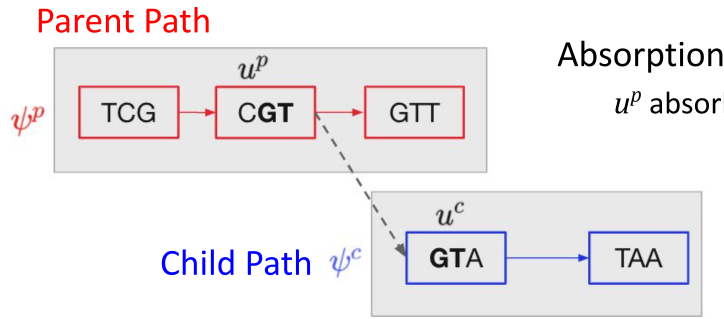


Absorption edge -----

u^p absorbs u^c (ψ^p absorbs ψ^c)

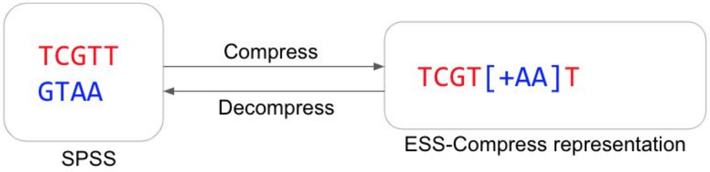


ESS-Compress representation



Absorption edge - - - - -

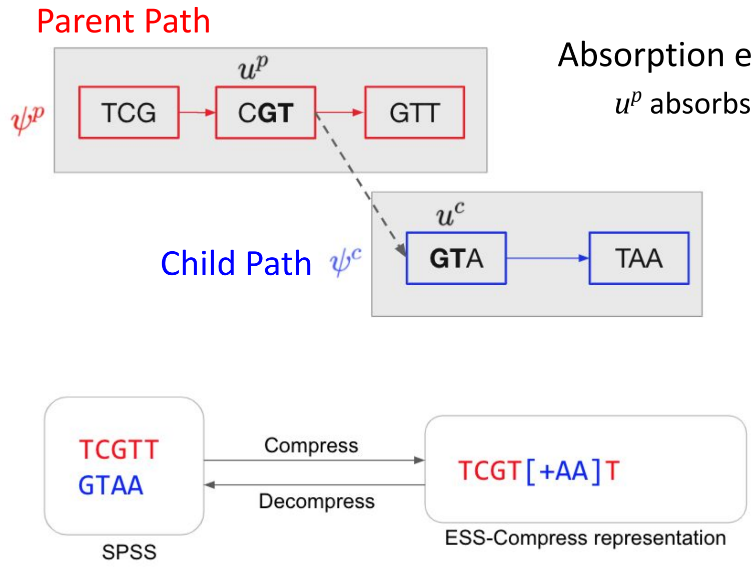
u^p absorbs u^c (ψ^p absorbs ψ^c)



Absorption process

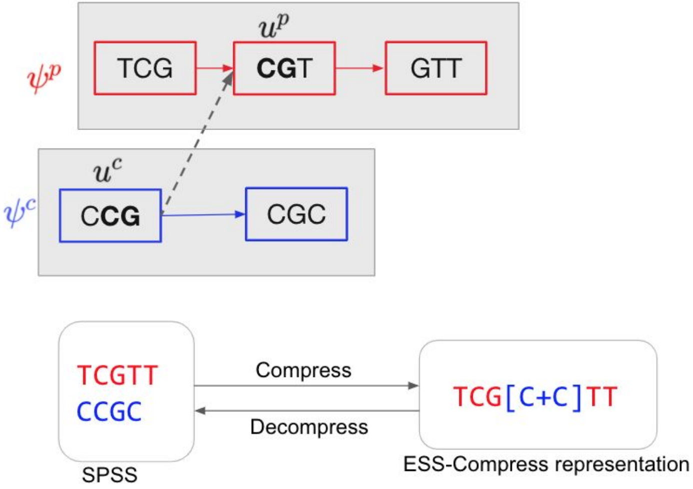
- Adds 3 extra characters: [, +,]
- Reduces (k-1) characters
- Overall (k-4) characters saved

ESS-Compress representation



Absorption edge -----

u^p absorbs u^c (ψ^p absorbs ψ^c)

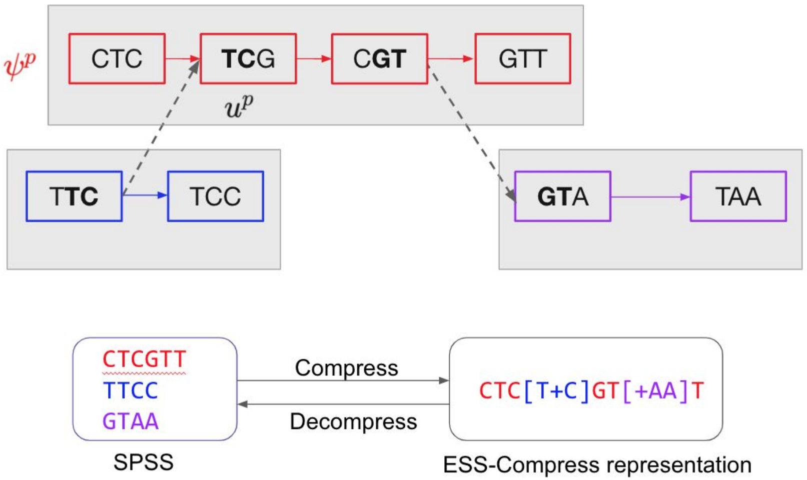


Absorption process

- Adds 3 extra characters: [, +,]
- Reduces (k-1) characters
- Overall (k-4) characters saved

ESS-Compress representation: more examples

One *path* absorbs multiple paths



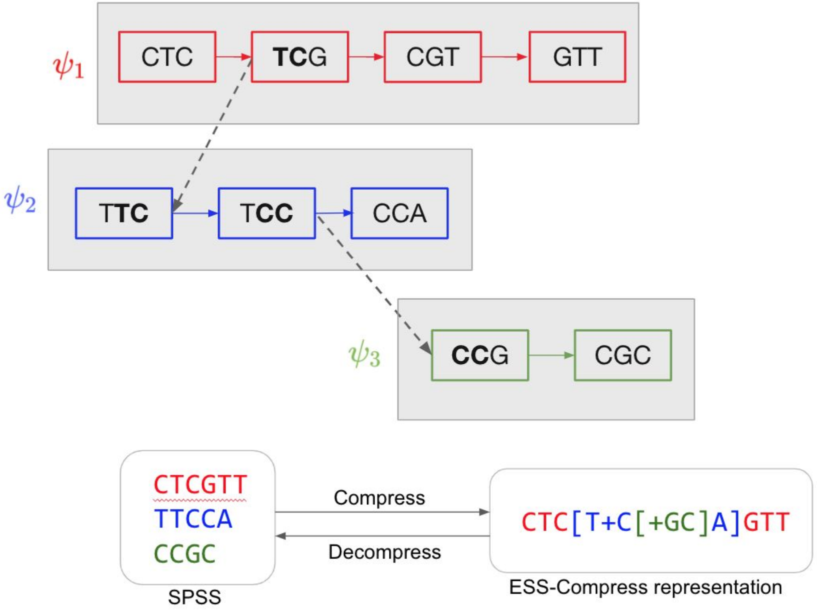
CTCGTT
TTCC
GTAA

SPSS

CTC[T+C]GT[+AA]T

ESS-Compress representation

Recursive absorption



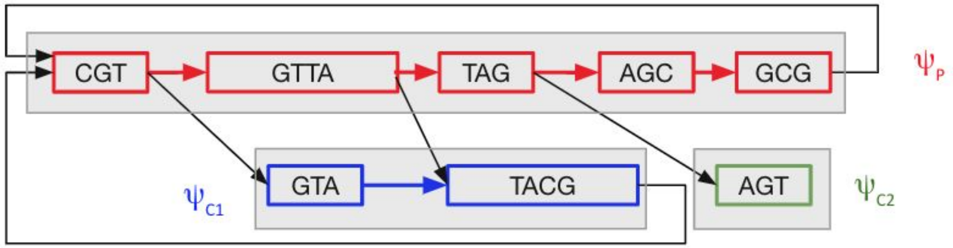
CTCGTT
TTCCA
CCGC

SPSS

CTC[T+C[+GC]A]GTT

ESS-Compress representation

Algorithm to compute ESS-Compress representation



CGTTAGCG
GTACG
AGT

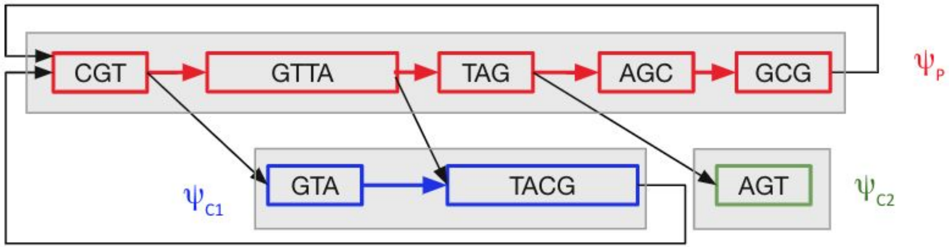
SPSS Representation

CGT[+ACG]TAG[+T]GCG

ESS-Compress Representation

Which absorption edges should be chosen? In what order?

Algorithm to compute ESS-Compress representation

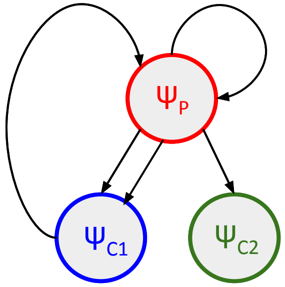


CGTTAGCG
GTACG
AGT

SPSS Representation

CGT[+ACG]TAG[+T]GCG

ESS-Compress Representation

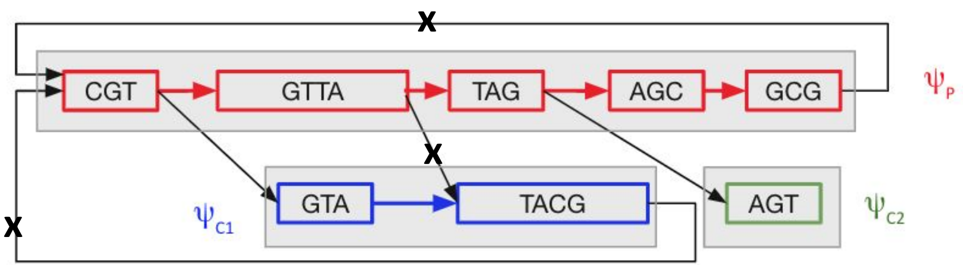


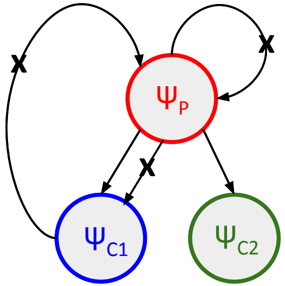
Absorption digraph

- Vertices are paths in compacted dBG
- Edge from path ψ_P to path ψ_C if
  - There is an absorption edge from ψ_P to ψ_C

Which absorption edges should be chosen? In what order?

Algorithm to compute ESS-Compress representation





CGTTAGCG
GTACG
AGT

SPSS Representation

CGT[+ACG]TAG[+T]GCG

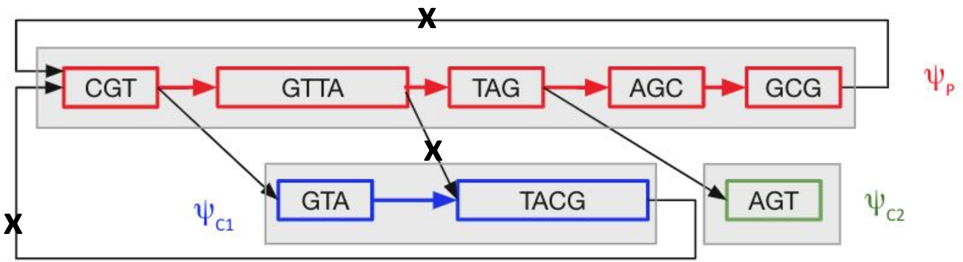
ESS-Compress Representation

Absorption digraph

- Vertices are paths in compacted dBG
- Edge from path ψ_P to path ψ_C if
  - There is an absorption edge from ψ_P to ψ_C

Which absorption edges should be chosen? In what order?

Algorithm to compute ESS-Compress representation

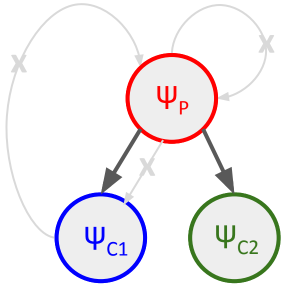


CGTTAGCG
GTACG
AGT

SPSS Representation

CGT[+ACG]TAG[+T]GCG

ESS-Compress Representation



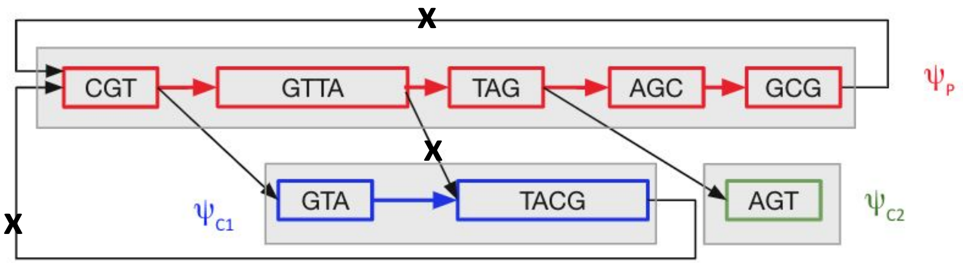
Absorption digraph

- Vertices are paths in compacted dBG
- Edge from path ψ_p to path ψ_c if
  - There is an absorption edge from ψ_p to ψ_c

Which absorption edges should be chosen? In what order?

- Compute ***edge-maximizing*** spanning out-forest
  - Analog of MST in directed graph

Algorithm to compute ESS-Compress representation

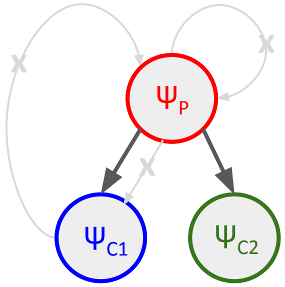


CGTTAGCG
GTACG
AGT

SPSS Representation

CGT[+ACG]TAG[+T]GCG

ESS-Compress Representation

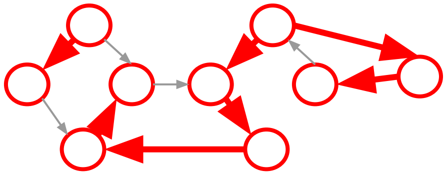


Absorption digraph

- Vertices are paths in compacted dBG
- Edge from path ψ_P to path ψ_C if
  - There is an absorption edge from ψ_P to ψ_C

Which absorption edges should be chosen? In what order?

- Compute ***edge-maximizing*** spanning out-forest
  - Analog of MST in directed graph
- We give an algorithm
  - DFS-based
  - linear time
  - returns out-forest with **maximal edges** and **minimal number of out-trees**

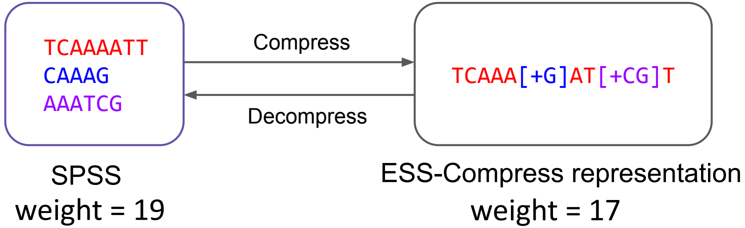


edge-maximizing spanning out-forest *in red*

Weight and lower bound of ESS-Compress representation

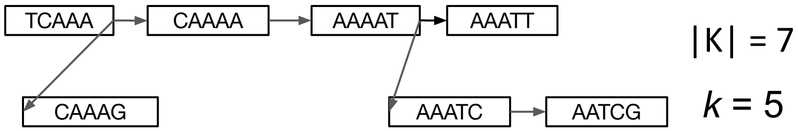
weight of ESS solution = $|K| + 3|\psi| + n_{sc}(k - 4)$

- $|K|$ = n. of k-mers
- $|\psi|$ = n. of paths in path cover
- n_{sc} = n. of source in strongly connected component metagraph



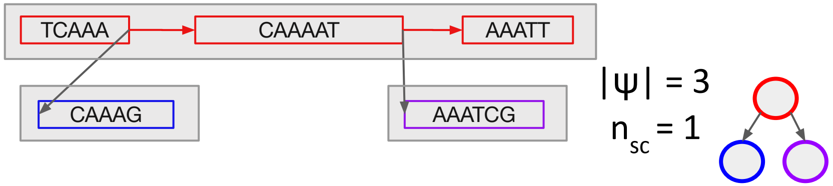
SPSS
weight = 19

ESS-Compress representation
weight = 17



$|K| = 7$

$k = 5$



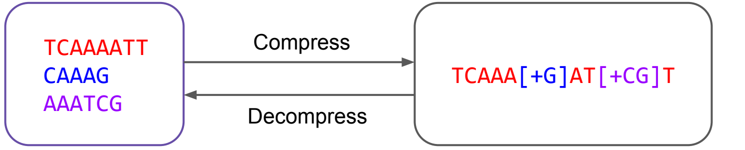
$|\psi| = 3$

$n_{sc} = 1$

Weight and lower bound of ESS-Compress representation

weight of ESS solution = $|K| + 3|\psi| + n_{sc}(k - 4)$

- $|K|$ = n. of k-mers
- $|\psi|$ = n. of paths in path cover
- n_{sc} = n. of source in strongly connected component metagraph


TCAAAATT
CAAAG
AAATCG

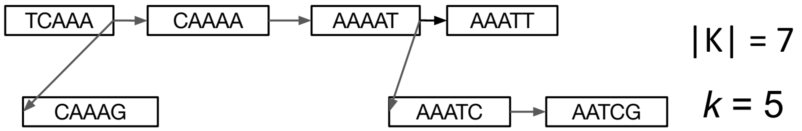
Compress

Decompress

TCAAA[+G]AT[+CG]T

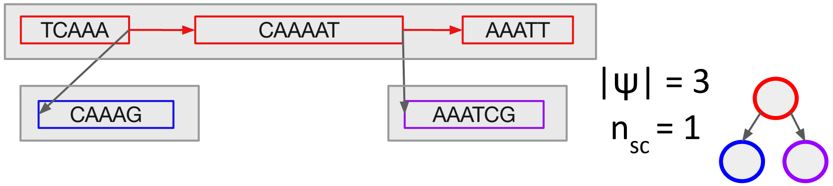
SPSS
weight = 19

ESS-Compress representation
weight = 17



$|K| = 7$

$k = 5$

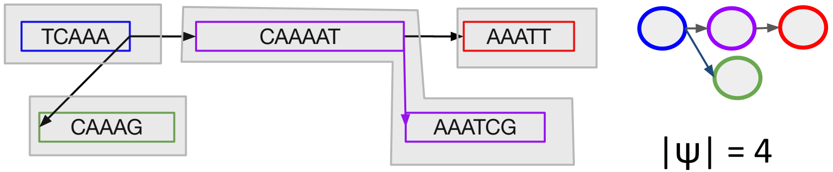


$|\psi| = 3$

$n_{sc} = 1$

Possible modifications within ESS compress framework

- choosing different starting path cover
- choose different edges as absorption edges

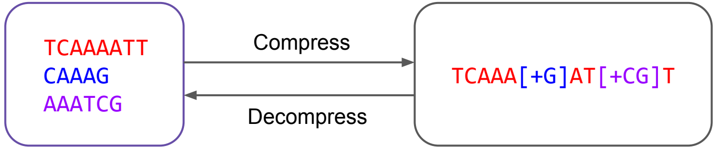


$|\psi| = 4$

Weight and lower bound of ESS-Compress representation

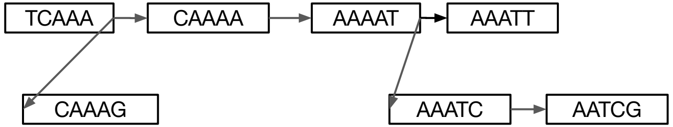
weight of ESS solution = $|K| + 3|\psi| + n_{sc}(k - 4)$

- $|K|$ = n. of k-mers
- $|\psi|$ = n. of paths in path cover
- n_{sc} = n. of source in strongly connected component metagraph



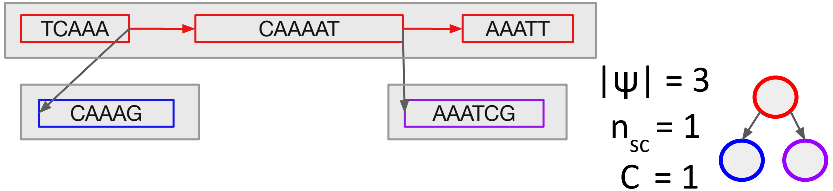
SPSS
weight = 19

ESS-Compress representation
weight = 17



$|K| = 7$

$k = 5$



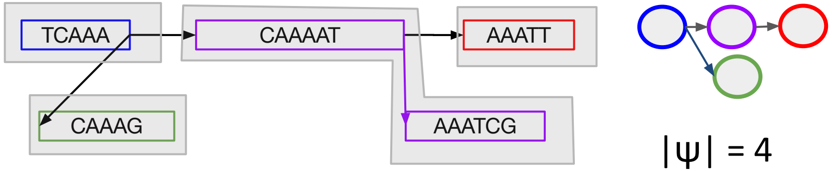
$|\psi| = 3$

$n_{sc} = 1$

$C = 1$

Possible modifications within ESS compress framework

- choosing different starting path cover
- choose different edges as absorption edges



$|\psi| = 4$

$n_{sc} = 1$

$C = 1$

Lower bound within this framework

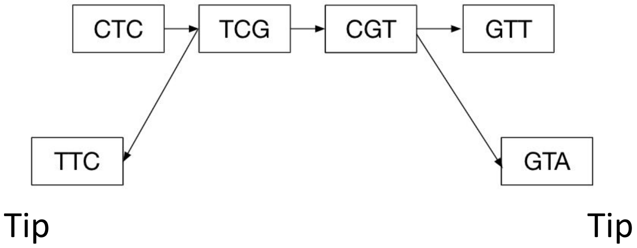
- weight >= $|K| + 3B + C(k - 4)$
  - B = lower bound of SPSS
  - C = # connected components in cdbG

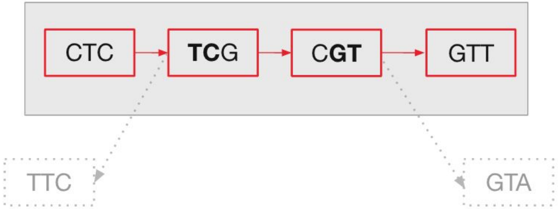
$C < n_{sc}$

$B < |\psi|$

ESS-Tip-Compress: a faster and simpler alternative

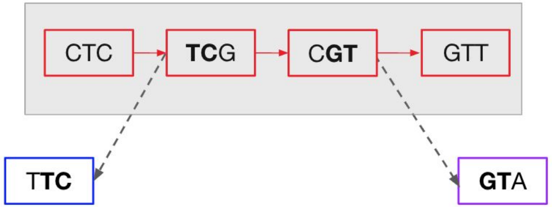
- ESS-compress can take a lot of memory because of recursion
- Observe that datasets have a large number of tips
- ESS-Tip-Compress
  - non-recursive
  - finds path cover of the graph minus tips
  - absorb tips into the path





CTCGTT

SPSS without tips

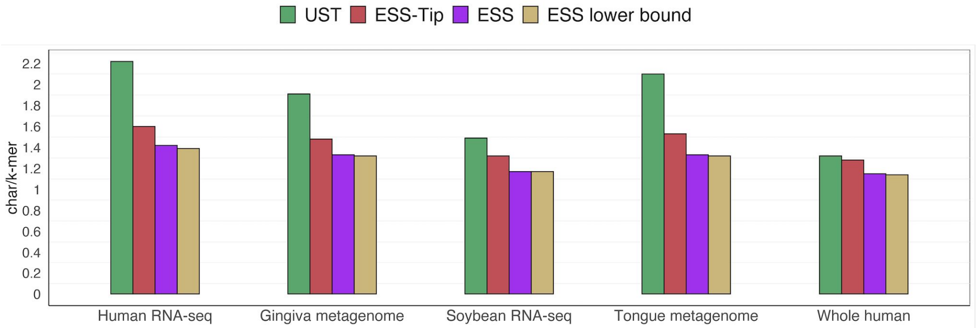


CTC{T{GT}A}T

ESS-Tip-Compress representation

RESULTS

Size of k-mer set representation



*101 mil. *101 mil. *111 mil. *165 mil. *2 bil.

* number of distinct 31-mers

ESS uses less characters

- **13-42%** better than UST
- **7-10%** better than ESS-Tip

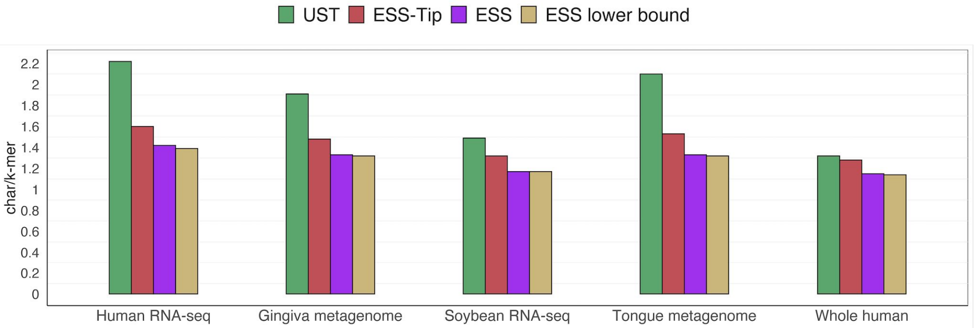
ESS-Tip

- more characters than ESS
- still better than UST

ESS is nearly optimal with respect to lower bound

- < **1.7%** gap

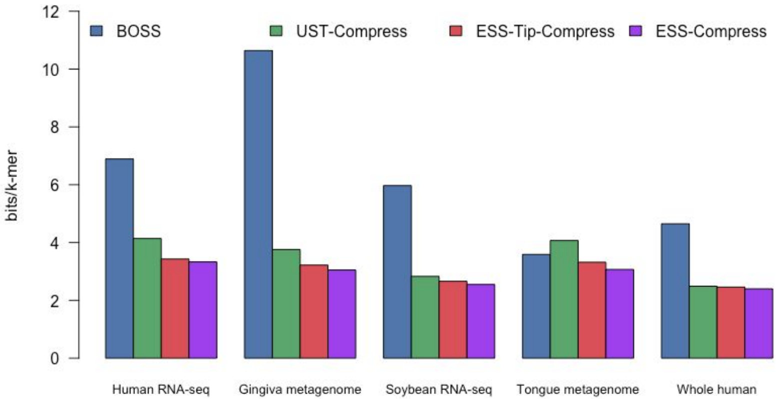
Size of k-mer set representation

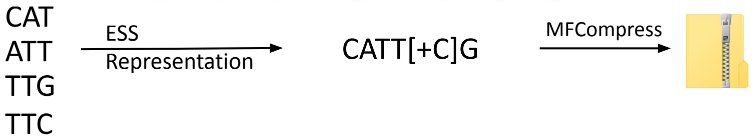


*101 mil. *101 mil. *111 mil. *165 mil. *2 bil.

* number of distinct 31-mers

Compressed File Size





ESS uses less characters

- **13-42%** better than UST
- **7-10%** better than ESS-Tip

ESS-Tip

- more characters than ESS
- still better than UST

ESS is nearly optimal with respect to lower bound

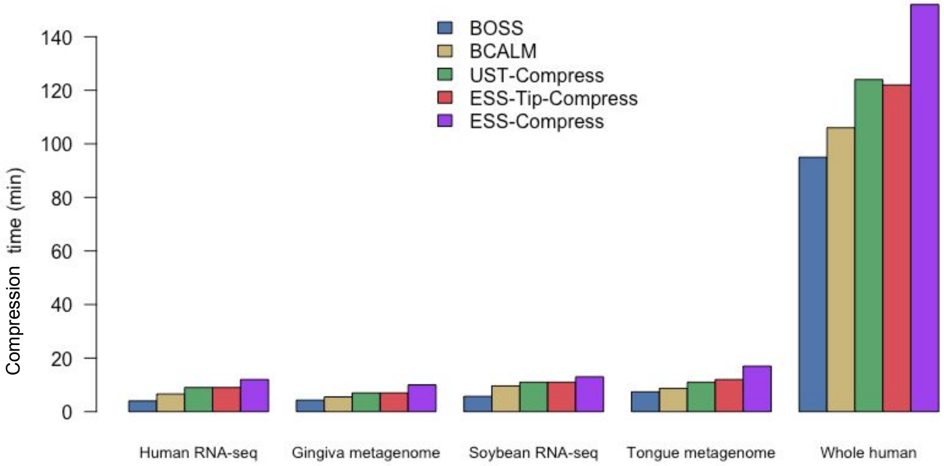
- < **1.7%** gap

ESS-Compress uses less space

- **6-27%** smaller than UST-Compress
- **7-10%** smaller than ESS-Tip-Compress
- **Order-of-magnitude** less than
  - MFC-compressed FASTA
  - Plaintext with one k-mer per line

Time and memory

- Compression time
- Compression memory:
  - For largest dataset, peak memory
    - ESS-Compress = 42 GB
    - ESS-Tip-Compress = 11 GB
  - For other datasets:
    - ESS-Compress < 10 GB
    - ESS-Tip-Compress < 3 GB
- Decompression memory:
  - ESS-Compress < 0.7 GB
  - ESS-Tip-Compress < 0.5 GB
- Decompression Time:
  - For large dataset: < 10 min
  - For others: < 2 min
- Advantage of ESS-Tip-Compress
  - Compression memory and time:
    - UST-Compress ≈ ESS-Tip-Compress ≪ ESS-Compress
  - Only **7-10%** worse than ESS-Compress in compression size
    - Reasonable trade-off



*101 mil. *101 mil. *111 mil. *165 mil. *2 bil.

* number of distinct 31-mers

Summary

- ESS-Compress reduces space
  - Order-of-magnitude smaller than MFC-compressed FASTA and plaintext with one k-mer per line
  - **6-27%** smaller than UST-Compress
  - Nearly optimal **(< 1.7% gap)** within its class.

- Efficient
  - Compression **< 20 minutes** for medium sized datasets
  - Decompression **< 1 minute**

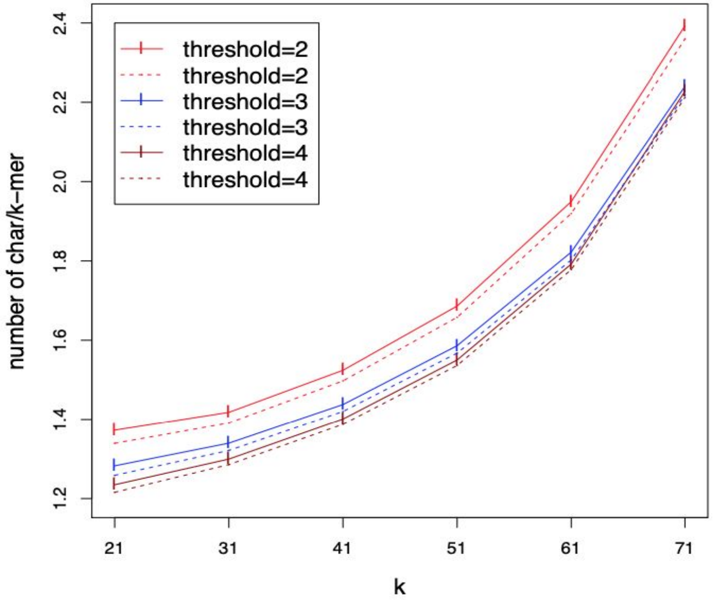
- ESS-Tip-Compress: a simpler alternative
  - Time and memory: UST-Compress **≈** ESS-Tip-Compress ≪ ESS-Compress
  - Compressed size: only **7-10%** worse than ESS-Compress

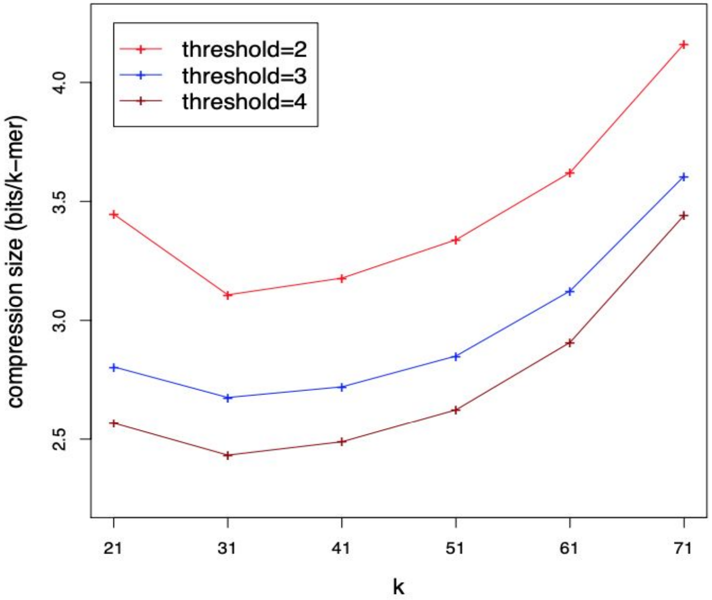
- Acknowledgments
  - NSF awards 1453527 and 1439057
  - NIH Computation, Bioinformatics, and Statistics (CBIOS) training program

- Software availability: github.com/medvedevgroup/ESSCompress

Supplementary Slides

Effect of varying *k* on compression performance on human RNA-seq data





Dashed lines represent empirical lower bound

- Weight of ESS-Compress closely matches lower bound (< 2.4% gap)
- Better compression when
  - *k* reduces
  - abundance threshold increases
  - Due to decrease in # connected components in dbG

B = lower bound of SPSS

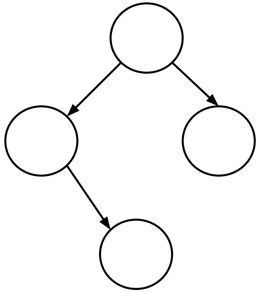
C = # connected components in cdbG

Lower bound of weight
= |K| + 3B + C(k - 4)

Spanning out-forest

An *out-tree* in a directed graph *D* is a subgraph where

- every vertex except a single root, has in-degree 1
- the underlying undirected graph is a tree.



out-tree

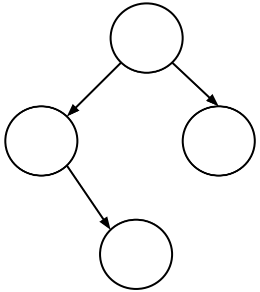
Spanning out-forest

An *out-tree* in a directed graph *D* is a subgraph where

- every vertex except a single root, has in-degree 1
- the underlying undirected graph is a tree.

An *out-forest* is a collection of vertex-disjoint out-trees.

- An out-forest is spanning if it covers all the vertices of D



out-tree

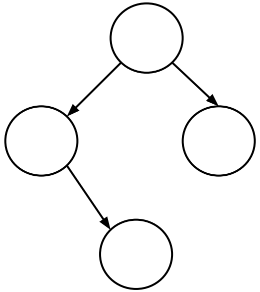
Spanning out-forest

An *out-tree* in a directed graph *D* is a subgraph where

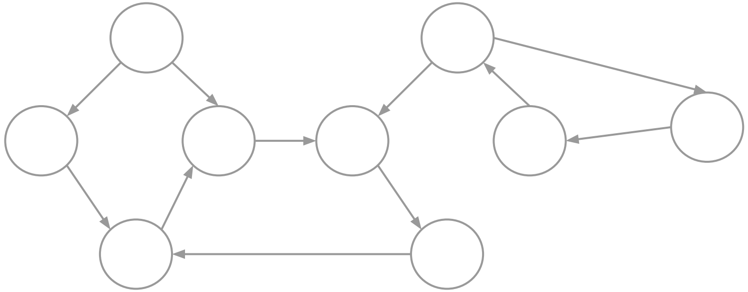
- every vertex except a single root, has in-degree 1
- the underlying undirected graph is a tree.

An *out-forest* is a collection of vertex-disjoint out-trees.

- An out-forest is spanning if it covers all the vertices of D



out-tree



Directed graph

Spanning out-forest

An *out-tree* in a directed graph *D* is a subgraph where

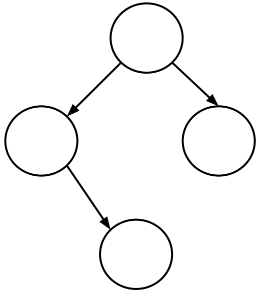
- every vertex except a single root, has in-degree 1
- the underlying undirected graph is a tree.

An *out-forest* is a collection of vertex-disjoint out-trees.

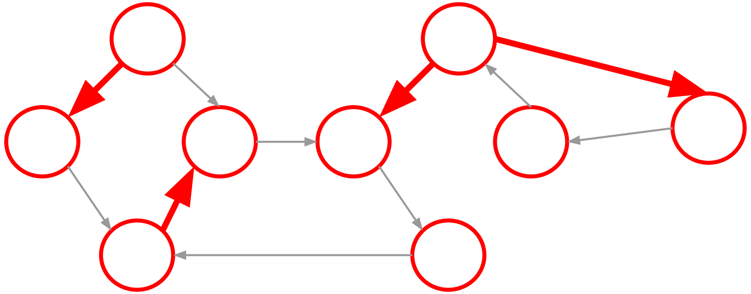
- An out-forest is spanning if it covers all the vertices of D

But this is not optimal:

- possible to increase number of edges



out-tree



Directed graph, with ***spanning out-forest*** *in red*

Spanning out-forest

An *out-tree* in a directed graph *D* is a subgraph where

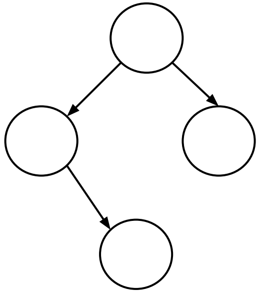
- every vertex except a single root, has in-degree 1
- the underlying undirected graph is a tree.

An *out-forest* is a collection of vertex-disjoint out-trees.

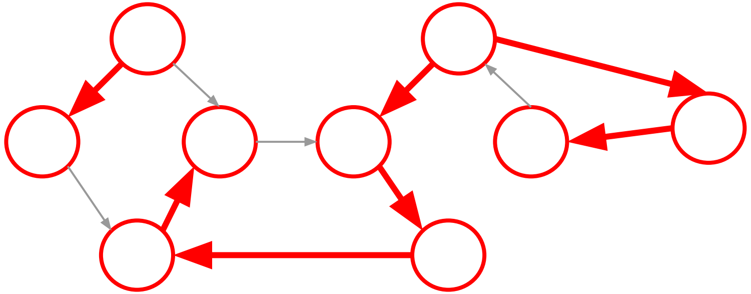
- An out-forest is spanning if it covers all the vertices of D

Optimal

- Maximal edges



out-tree



Directed graph, with ***spanning out-forest*** *in red*

We give an algorithm to find an optimal spanning out-forest

- We prove that it gives the **maximal edges** and **minimal number of trees**
  - a specific instance of the maximum weight out-forest problem

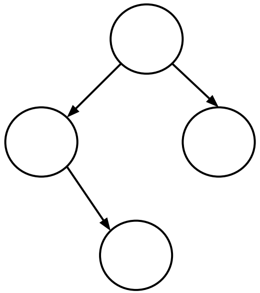
Spanning out-forest

An *out-tree* in a directed graph *D* is a subgraph where

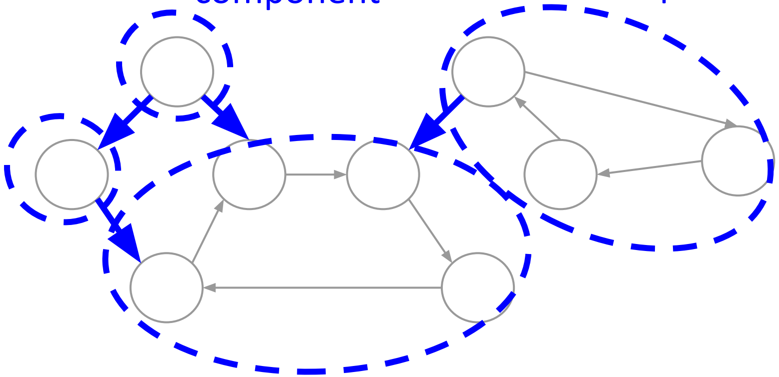
- every vertex except a single root, has in-degree 1
- the underlying undirected graph is a tree.

An *out-forest* is a collection of vertex-disjoint out-trees.

- An out-forest is spanning if it covers all the vertices of D



out-tree



Algorithm to find edge-maximizing spanning out-forest

We give an algorithm to find an optimal spanning out-forest

- We prove that it gives the **maximal edges** and **minimal number of trees**
  - a specific instance of the maximum weight out-forest problem

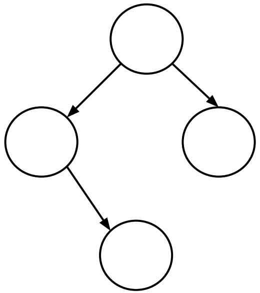
Spanning out-forest

An *out-tree* in a directed graph *D* is a subgraph where

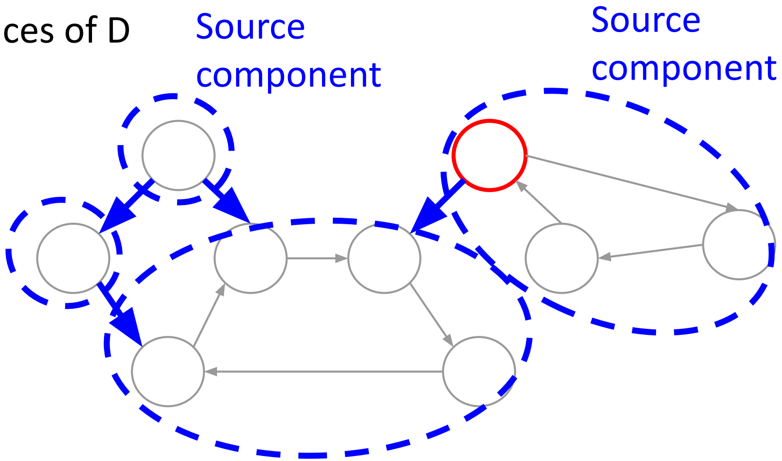
- every vertex except a single root, has in-degree 1
- the underlying undirected graph is a tree.

An *out-forest* is a collection of vertex-disjoint out-trees.

- An out-forest is spanning if it covers all the vertices of D



out-tree



Algorithm to find edge-maximizing spanning out-forest

We give an algorithm to find an optimal spanning out-forest

- We prove that it gives the **maximal edges** and **minimal number of trees**
  - a specific instance of the maximum weight out-forest problem

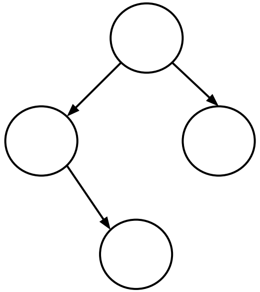
Spanning out-forest

An *out-tree* in a directed graph *D* is a subgraph where

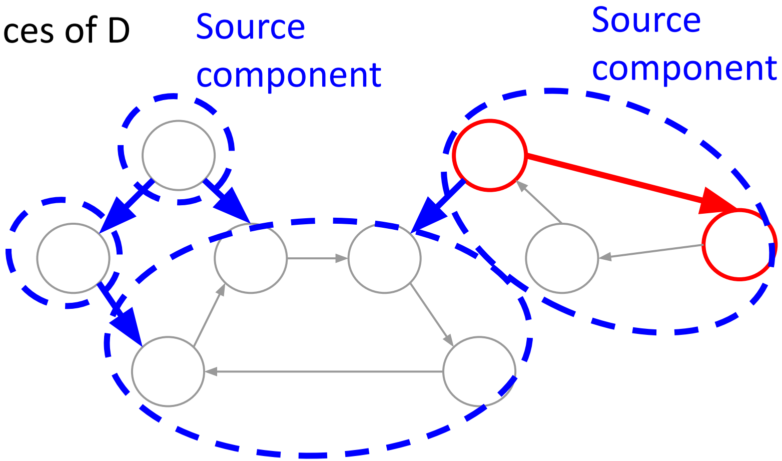
- every vertex except a single root, has in-degree 1
- the underlying undirected graph is a tree.

An *out-forest* is a collection of vertex-disjoint out-trees.

- An out-forest is spanning if it covers all the vertices of D



out-tree



Algorithm to find edge-maximizing spanning out-forest

We give an algorithm to find an optimal spanning out-forest

- We prove that it gives the **maximal edges** and **minimal number of trees**
  - a specific instance of the maximum weight out-forest problem

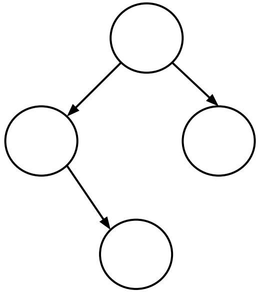
Spanning out-forest

An *out-tree* in a directed graph *D* is a subgraph where

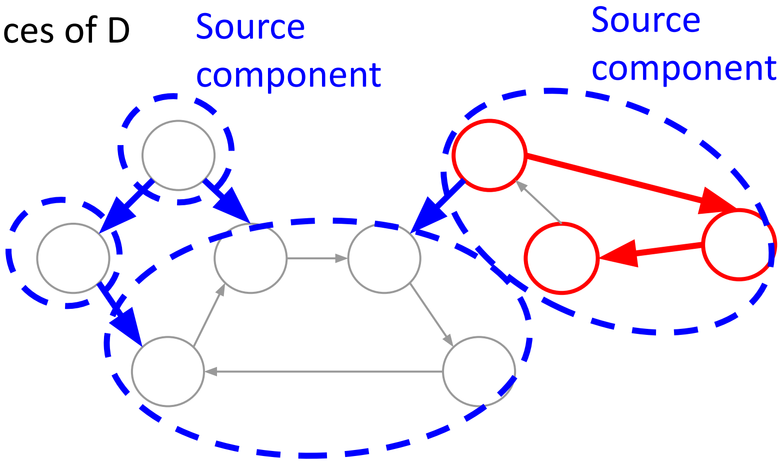
- every vertex except a single root, has in-degree 1
- the underlying undirected graph is a tree.

An *out-forest* is a collection of vertex-disjoint out-trees.

- An out-forest is spanning if it covers all the vertices of D



out-tree



Algorithm to find edge-maximizing spanning out-forest

We give an algorithm to find an optimal spanning out-forest

- We prove that it gives the **maximal edges** and **minimal number of trees**
  - a specific instance of the maximum weight out-forest problem

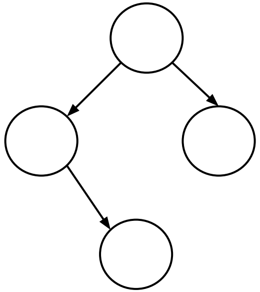
Spanning out-forest

An *out-tree* in a directed graph *D* is a subgraph where

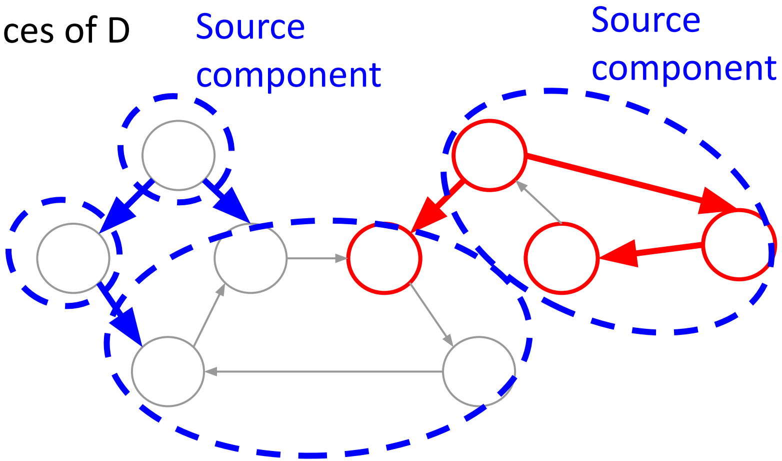
- every vertex except a single root, has in-degree 1
- the underlying undirected graph is a tree.

An *out-forest* is a collection of vertex-disjoint out-trees.

- An out-forest is spanning if it covers all the vertices of D



out-tree



Algorithm to find edge-maximizing spanning out-forest

We give an algorithm to find an optimal spanning out-forest

- We prove that it gives the **maximal edges** and **minimal number of trees**
  - a specific instance of the maximum weight out-forest problem

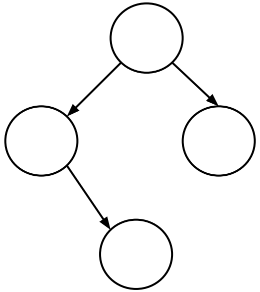
Spanning out-forest

An *out-tree* in a directed graph *D* is a subgraph where

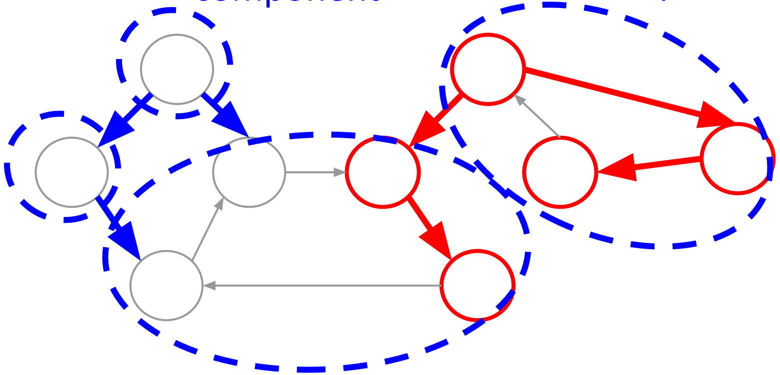
- every vertex except a single root, has in-degree 1
- the underlying undirected graph is a tree.

An *out-forest* is a collection of vertex-disjoint out-trees.

- An out-forest is spanning if it covers all the vertices of D



out-tree



Algorithm to find edge-maximizing spanning out-forest

We give an algorithm to find an optimal spanning out-forest

- We prove that it gives the **maximal edges** and **minimal number of trees**
  - a specific instance of the maximum weight out-forest problem

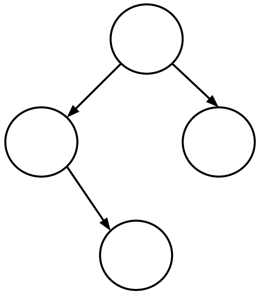
Spanning out-forest

An *out-tree* in a directed graph *D* is a subgraph where

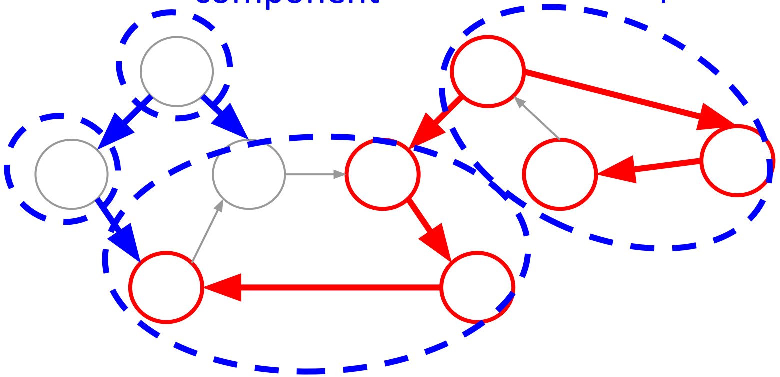
- every vertex except a single root, has in-degree 1
- the underlying undirected graph is a tree.

An *out-forest* is a collection of vertex-disjoint out-trees.

- An out-forest is spanning if it covers all the vertices of D



out-tree



Algorithm to find edge-maximizing spanning out-forest

We give an algorithm to find an optimal spanning out-forest

- We prove that it gives the **maximal edges** and **minimal number of trees**
  - a specific instance of the maximum weight out-forest problem

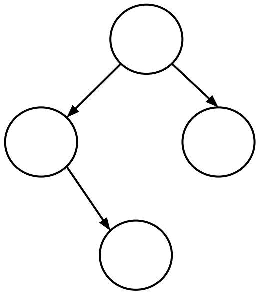
Spanning out-forest

An *out-tree* in a directed graph *D* is a subgraph where

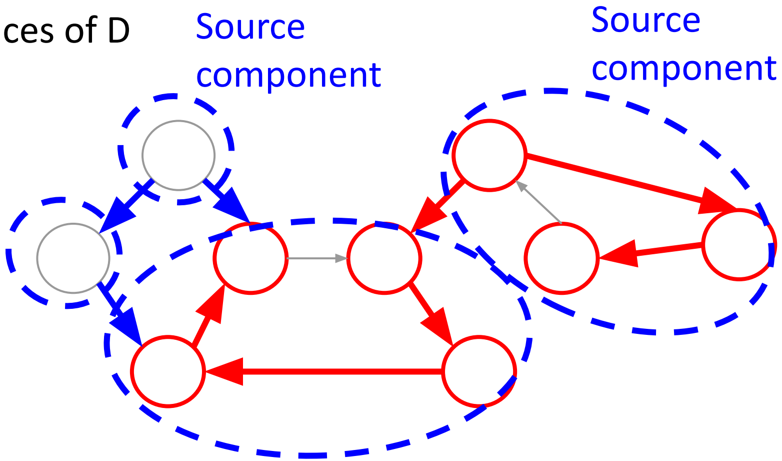
- every vertex except a single root, has in-degree 1
- the underlying undirected graph is a tree.

An *out-forest* is a collection of vertex-disjoint out-trees.

- An out-forest is spanning if it covers all the vertices of D



out-tree



Algorithm to find edge-maximizing spanning out-forest

We give an algorithm to find an optimal spanning out-forest

- We prove that it gives the **maximal edges** and **minimal number of trees**
  - a specific instance of the maximum weight out-forest problem

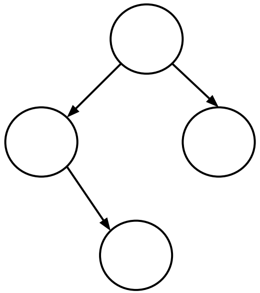
Spanning out-forest

An *out-tree* in a directed graph *D* is a subgraph where

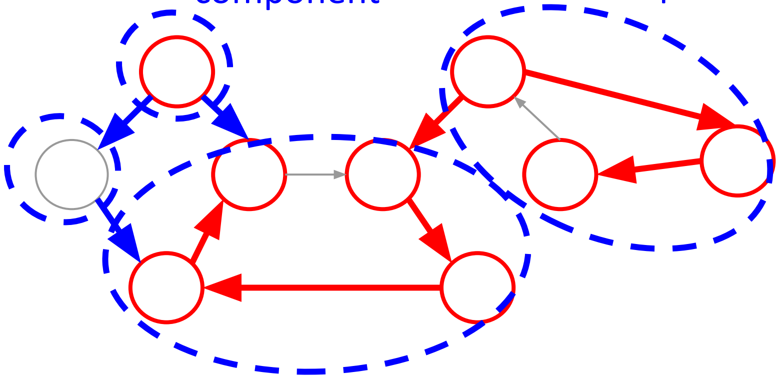
- every vertex except a single root, has in-degree 1
- the underlying undirected graph is a tree.

An *out-forest* is a collection of vertex-disjoint out-trees.

- An out-forest is spanning if it covers all the vertices of D



out-tree



Algorithm to find edge-maximizing spanning out-forest

We give an algorithm to find an optimal spanning out-forest

- We prove that it gives the **maximal edges** and **minimal number of trees**
  - a specific instance of the maximum weight out-forest problem

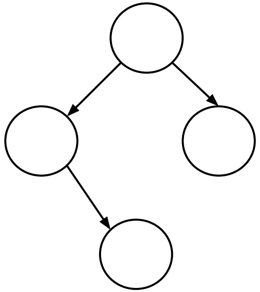
Spanning out-forest

An *out-tree* in a directed graph *D* is a subgraph where

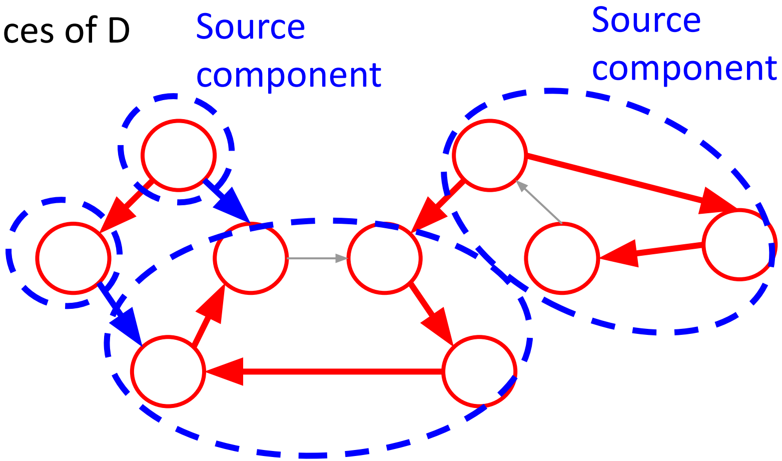
- every vertex except a single root, has in-degree 1
- the underlying undirected graph is a tree.

An *out-forest* is a collection of vertex-disjoint out-trees.

- An out-forest is spanning if it covers all the vertices of D



out-tree



Algorithm to find edge-maximizing spanning out-forest

We give an algorithm to find an optimal spanning out-forest

- We prove that it gives the **maximal edges** and **minimal number of trees**
  - a specific instance of the maximum weight out-forest problem