Superstrings: graphs, greedy algorithms and assembly

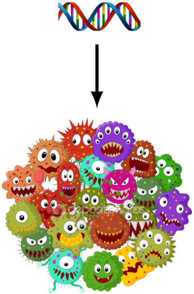
B. Cazaux and E. Rivals

* LIRMM & IBC, Montpellier

May 13, 2019

Importance of a genome sequence

Importance of a genome sequence



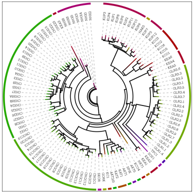
Sample analysis

Importance of a genome sequence

Species identification

Sample analysis

Importance of a genome sequence



Infer evolutionary relationships

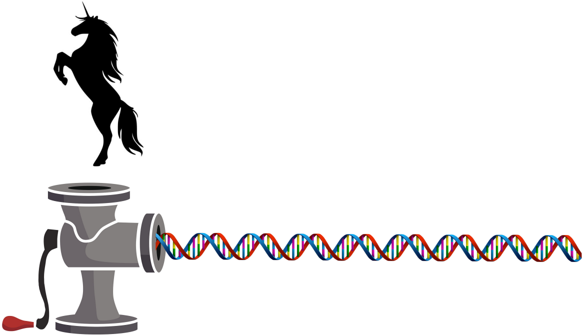
Species identification

Sample analysis

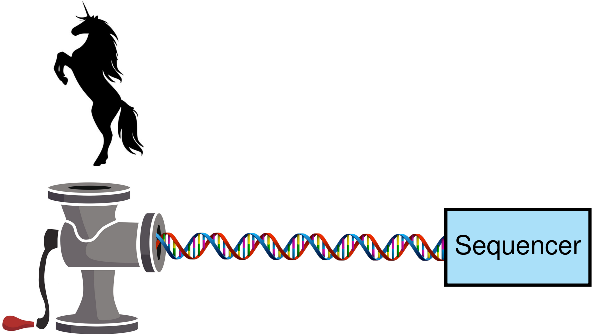
Genome shotgun sequencing

Genome shotgun sequencing

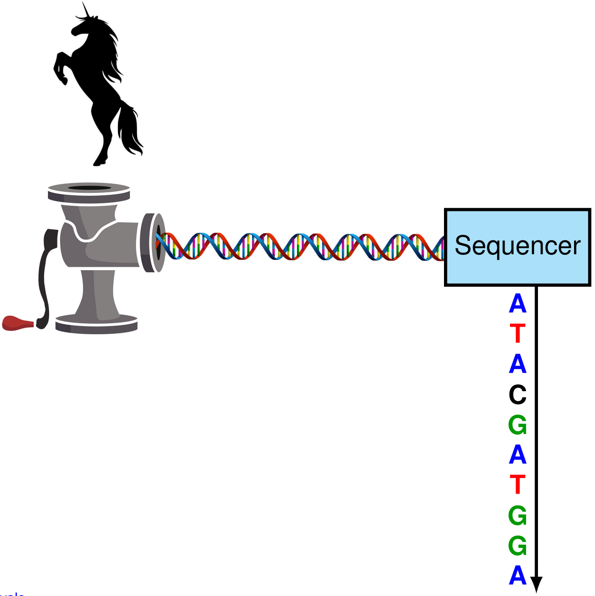
Genome shotgun sequencing



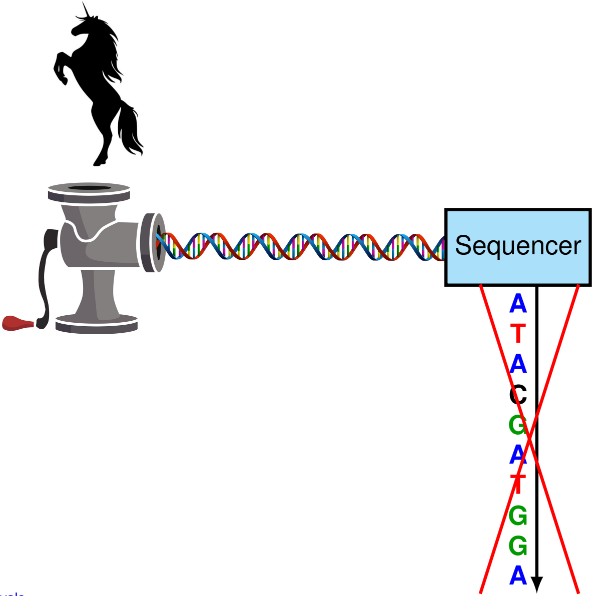
Genome shotgun sequencing



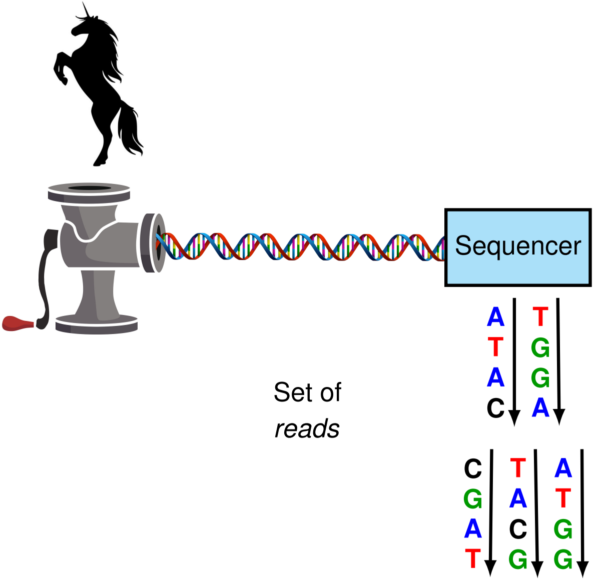
Genome shotgun sequencing



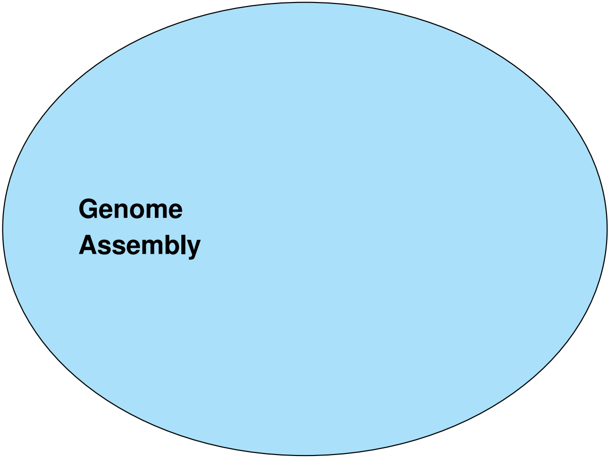
Genome shotgun sequencing



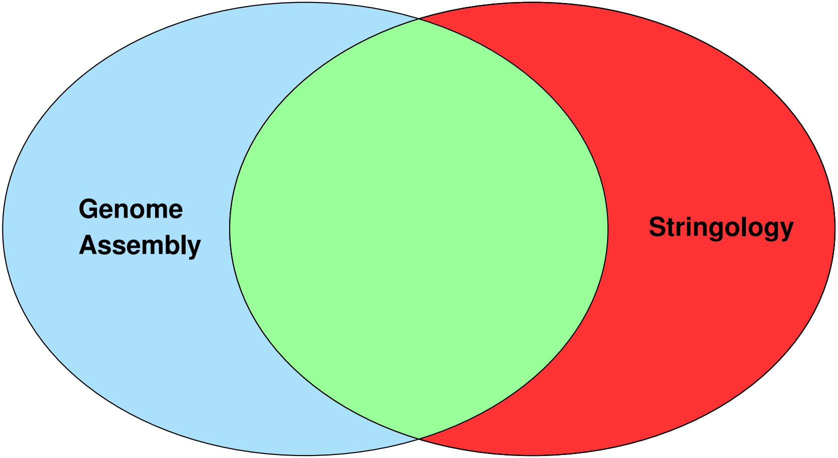
Genome shotgun sequencing



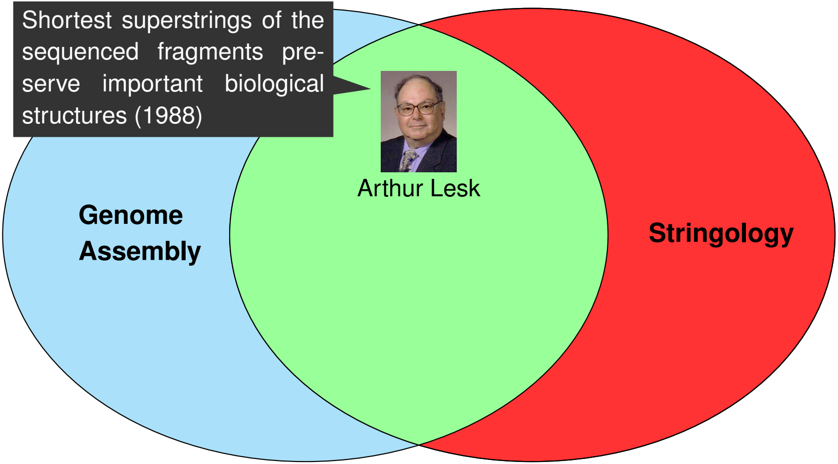
Genome assembly and shortest superstring



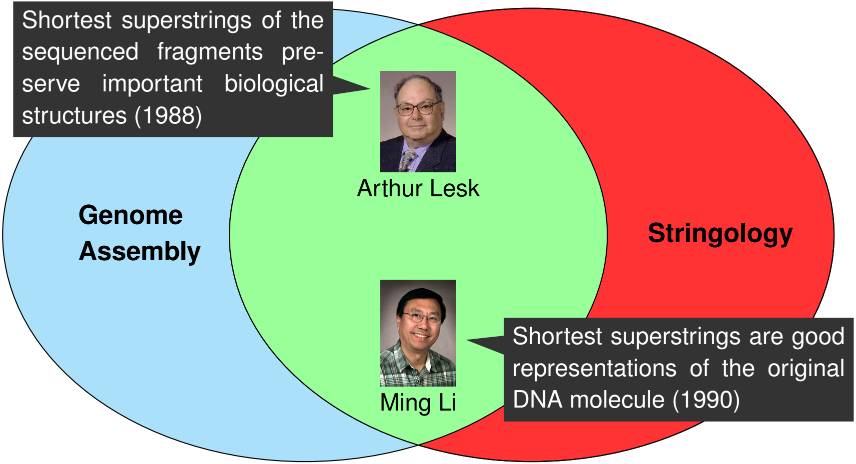
Genome assembly and shortest superstring



Genome assembly and shortest superstring



Genome assembly and shortest superstring



Applications of superstrings

Multiple applications in diverse domains

- DNA assembly [Gingeras 1979, Peltola 1982]
- text compression [Storer 1988]
- job scheduling [Middendorf 1998]
- vaccine design [Martinez 2015]

Review mentioning other applications [Gevezes, Pitsoulis, 2011].

Strings and superstrings: Basic definitions

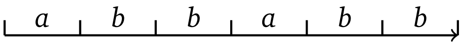
Vocable regarding strings or sequences

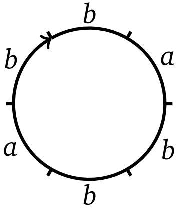
- Words = Strings = Sequence
- Sequence: ordered sequence of letters from alphabet
- We consider finite strings over an alphabet Σ
- and denote by $|v|$ the length of a string v.
- Substring = sequence in any interval in a string

Example

cde is a substring of *abcdeaeab* but not of *abcaedeae*

Linear and Cyclic words





Overlaps

Definition

Let *w* a string.

- a **substring** of *w* is a string included in *w*,
- a **prefix** of *w* is a substring which begins *w*
- a **suffix** is a substring which ends *w*.
- an **overlap** from *w* over *v* is a suffix of *w* that is also a prefix of *v*.

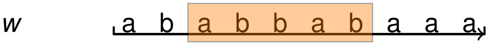
w a b a b b a b a a a

Overlaps

Definition

Let *w* a string.

- a **substring** of *w* is a string included in *w*,
- a **prefix** of *w* is a substring which begins *w*
- a **suffix** is a substring which ends *w*.
- an **overlap** from *w* over *v* is a suffix of *w* that is also a prefix of *v*.



Definition

Let *w* a string.

- a **substring** of *w* is a string included in *w*,
- a **prefix** of *w* is a substring which begins *w*
- a **suffix** is a substring which ends *w*.
- an **overlap** from *w* over *v* is a suffix of *w* that is also a prefix of *v*.

w a b a b b a b a a a

Overlaps

Definition

Let *w* a string.

- a **substring** of *w* is a string included in *w*,
- a **prefix** of *w* is a substring which begins *w*
- a **suffix** is a substring which ends *w*.
- an **overlap** from *w* over *v* is a suffix of *w* that is also a prefix of *v*.

w a b a b b a b a a a

Overlaps

Definition

Let *w* a string.

- a **substring** of *w* is a string included in *w*,
- a **prefix** of *w* is a substring which begins *w*
- a **suffix** is a substring which ends *w*.
- an **overlap** from *w* over *v* is a suffix of *w* that is also a prefix of *v*.

w a b a b b a b a a a

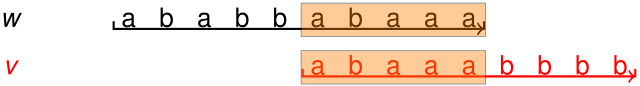
v a b a a a b b b b

Overlaps

Definition

Let *w* a string.

- a **substring** of *w* is a string included in *w*,
- a **prefix** of *w* is a substring which begins *w*
- a **suffix** is a substring which ends *w*.
- an **overlap** from *w* over *v* is a suffix of *w* that is also a prefix of *v*.

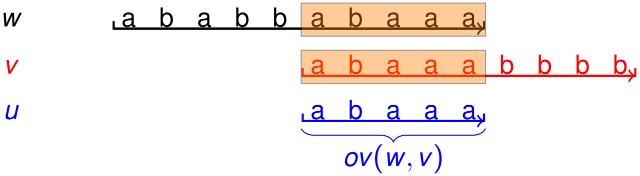


Overlaps

Definition

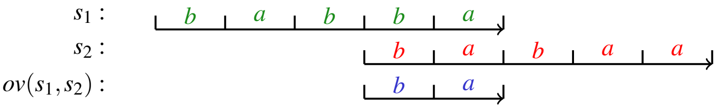
Let *w* a string.

- a **substring** of *w* is a string included in *w*,
- a **prefix** of *w* is a substring which begins *w*
- a **suffix** is a substring which ends *w*.
- an **overlap** from *w* over *v* is a suffix of *w* that is also a prefix of *v*.



Example (Maximum overlap between two strings)

Let strings $s_1 :=$ babba and $s_2 :=$ babaa.



s_1 overlaps s_2 by two characters

overlaps are not symmetric

Input of superstring problems

Throughout this article, the input is $P := \{s_1, \ldots, s_n\}$ a set of words.

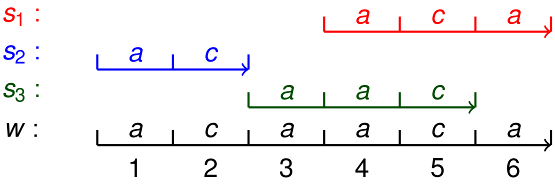
Without loss of generality, P is assumed to be substring free:

no word of P is substring of another word of P.

Let us denote the norm of P by $\|P\| := \sum_1^n |s_i|$.

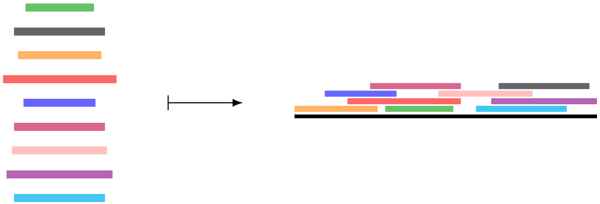
Definition Superstring

Let $P = \{s_1, s_2, \ldots, s_p\}$ be a set of strings. A *superstring* of P is a string w such that any s_i is a substring of w.



Definition Shortest Linear Superstring problem (SLS)

Input: P a set of finite strings over an alphabet Σ
Output: w a linear superstring of P of minimal length.



State of the art

Problem: Shortest Linear Superstrings problem (SLS)

- NP-hard [Gallant 1980]
- difficult to approximate [Blum et al. 1991]
- best known approximation ratio $2 + \frac{11}{30}$ [Paluch 2015]

Measures of approximation

One can consider two measures of approximation for SLS and its variants:

- the length of the output superstring w,

  which has to be minimised.
- the compression of P obtained with the superstring w, that is:

$$\sum_{i=1..p} |s_i| - |w|$$

  which has to be maximised.

Figure 2: Consider the $P := \{abba, bbabab, ababa, babaa\}$. (a) The string *abbababaa* is a superstring of P of length 9; the figure shows the order of the word of P in the superstring. (b) The sum of the overlaps between adjacent words in *abbababaa* equals $\|P\| - |abbababaa| = 11$.

Greedy algorithm

Greedy algorithm for SLS

A simple heuristic algorithm

- that builds a superstring
- by merging a pair of words with the largest maximum overlap
- introduced by [Tarhio, Ukkonen 1988]
- whose compression ratio can be guaranted,
- whose superstring ratio can also be guaranted
- and has a known lower bound.

greedy algorithm

Algorithm 1: **greedy** for Shortest Linear Superstring

Input: P a set of linear words.; **Output**: w a superstring of P;

while *P is not empty* **do**

  u and v two elements of P having the longest overlap ($u \neq v$)
  w is the merge of u and v
  $P := P \setminus \{u, v\}$
  if *P is empty (i.e. w is a superstring)* **then return** w **else**
  $P := P \cup \{w\}$

Theorem 1

Let P be a set of words. For any superstring w output by **greedy** there exists σ a permutation of P such that $w = \text{merge}(P, \sigma)$.

Example of greedy algorithm for SLS

a a b

a b b a

a b a a

a b a b b

Example of greedy algorithm for SLS

$$|ov(ababb, abba)| = |abb| = 3$$

aab

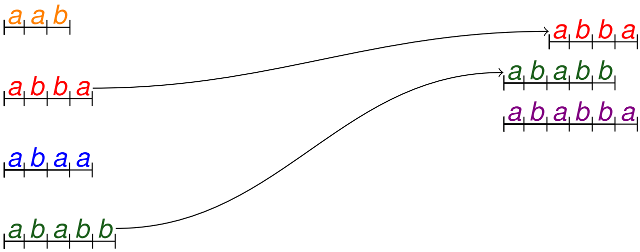
abba

abaa

ababb

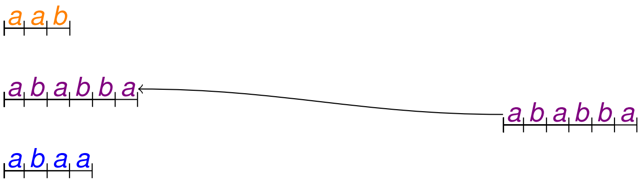
Example of greedy algorithm for SLS

$$|ov(ababb, abba)| = |abb| = 3$$



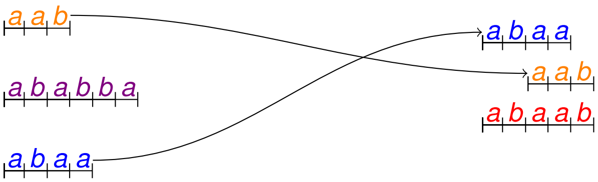
Example of greedy algorithm for SLS

$$|ov(ababb, abba)| = |abb| = 3$$



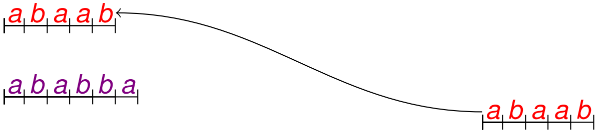
Example of greedy algorithm for SLS

$$|ov(abaa, aab)| = |aa| = 2$$



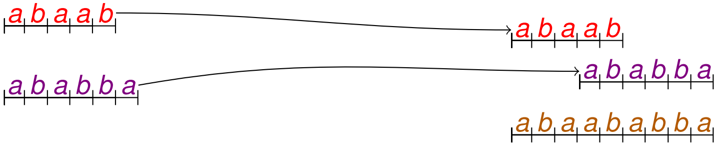
Example of greedy algorithm for SLS

$$|ov(abaa, aab)| = |aa| = 2$$



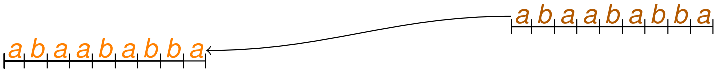
Example of greedy algorithm for SLS

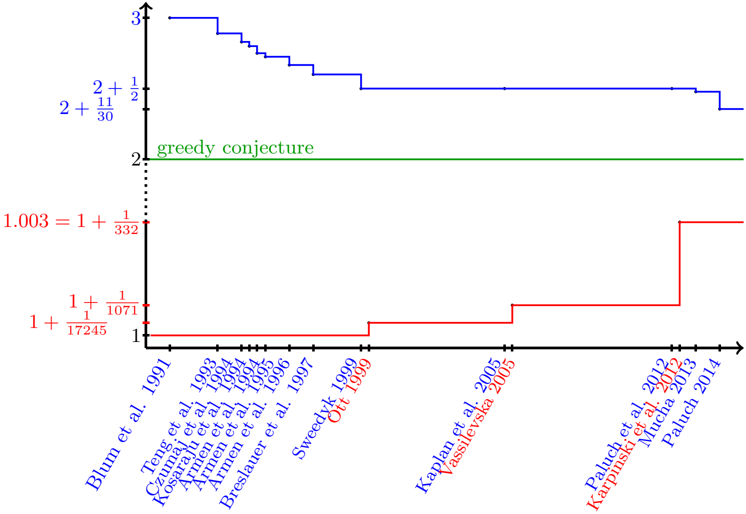
$$|ov(abaab, ababba)| = |ab| = 2$$



Example of greedy algorithm for SLS

$$|ov(abaab, ababba)| = |ab| = 2$$

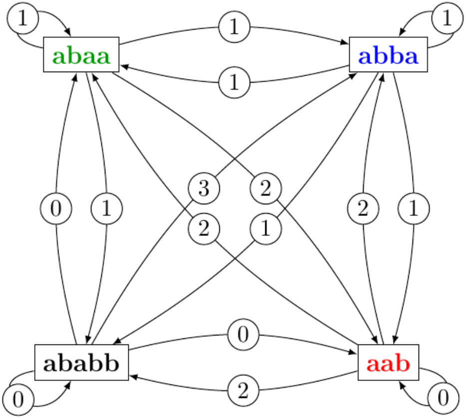



3
$2+\frac{1}{2}$
$2+\frac{11}{30}$
greedy conjecture
2
$1.003 = 1+\frac{1}{332}$
$1+\frac{1}{1071}$
$1+\frac{1}{17245}$
1
Blum et al. 1991
Teng et al. 1993
Czumaj et al. 1994
Kosaraju et al. 1994
Armen et al. 1994
Armen et al. 1995
Armen et al. 1996
Breslauer et al. 1997
Sweedyk 1999
Ott 1999
Kaplan et al. 2005
Vassilevska 2005
Paluch et al. 2012
Karpinski et al. 2012
Mucha 2013
Paluch 2014

Overlap Graph

Overlap Graph for a set of words

Consider the set
$P :=$
$\{$*abaa*, *abba*, *ababb*, *aab*$\}$



The Overlap Graph (OG) is applied in shortest superstring problems, DNA assembly, and other applications [Gevezes, Pitsoulis, 2011]

SLS and Max Weighted Hamiltonian path

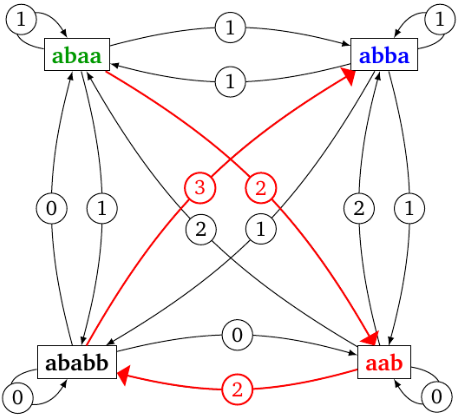
Theorem 2

Solving SLS of an instance P is equivalent to finding a Max Weighted Hamiltonian path on the Overlap Graph of P.

Idea:

- All words are contained,
- pairs of words are merged using their $ov(.,.)$
- the MWHP ensures the compression is maximised.

Ex. Max Weighted Hamiltonian path



Let $P := \{$**abaa**, **abba**, **ababb**, **aab**$\}$.

Optimal solution: $w = abaa\,b\,abb\,a = abaababba$

Overlap graph

- Quadratic number of arcs / weights to compute
- Computing the weights requires to solve
  the so-called All Pairs Suffix Prefix overlaps problem (APSP)
- Optimal time algorithms for APSP by
  [Gusfield et al 1992] and others [Lim, Park 2017] or [Tustumi et al. 2016].
- Useful information are difficult to get in the OG

We propose an alternative to the OG,
called the **Hierarchical Overlap Graph**
and an algorithm to build it.

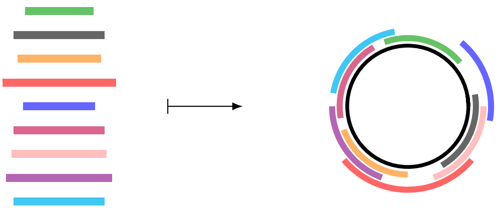
SLS and its variants

Shortest Cyclic Superstrings problem

Problem: Shortest Cyclic Superstrings problem (SCS)

Input: A set of linear words P

Output: w a cyclic superstring of P of minimal length.

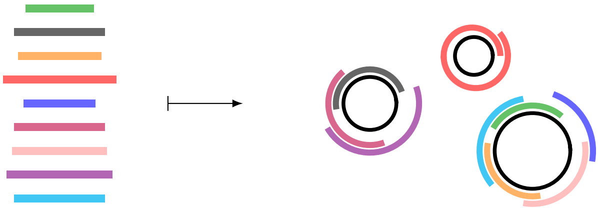


Shortest Cyclic Cover of Strings problem

Problem: Shortest Cyclic Cover of Strings problem (SCCS)

Input: A set of linear words P

Output: A set of minimum cyclic words C, such that $\forall s \in P$, $\exists c \in C$, such that s is a substring of c (minimum for the sum of the length of the words of C).



State of the art

Problem: Shortest Linear Superstring (SLS)

- NP-hard [Gallant 1980]
- difficult to approximate [Blum et al. 1991]
- best known approximation ratio $2 + \frac{11}{30}$ [Paluch 2015]

Problem: Shortest Cyclic Superstring (SCS)

- NP-hard [Cazaux, thesis 2016]
- difficult to approximate ????
- best known approximation ratio ????

Problem: Shortest Cyclic Cover of String (SCCS)

- Polynomial time for SCC in graph [Papadimitriou & Stieglitz]
- Linear [Cazaux & R., JDA, 2016]

Length of optimal solutions of SLS, SCS, SCCS

Theorem 3

Let P be an instance of SLS, SCS, SCCS.
We have

$$|opt(SLS)| \geq |opt(SCS)| \geq |opt(SCCS)|.$$

Greedy algorithm for SCCS

Algorithm 2: `greedy` for Shortest Cyclic Cover of Strings

Input: P a set of linear words.;

Output: S a set of cyclic strings that cover P;

$S := \emptyset$

while *P is not empty* **do**

  u and v two elements of P having the longest overlap (u can be equal to v)

  w is the merge of u and v

  $P := P \setminus \{u, v\}$

  if $u = v$ *(i.e. w is a cyclic string)* **then** $S := S \cup \{w\}$ **else**

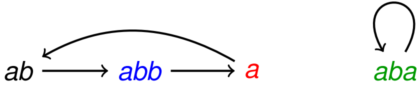
  $P := P \cup \{w\}$

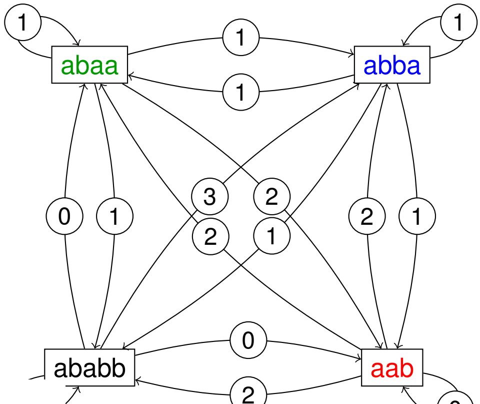
return S

Merging words from a permutation

$$P = \{ababb, \color{red}{aab}, \color{blue}{abba}, \color{green}{abaa}\} \quad + \quad \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 2 & 4 \end{pmatrix} = \sigma_1$$

$CC(P, \sigma_1) =$

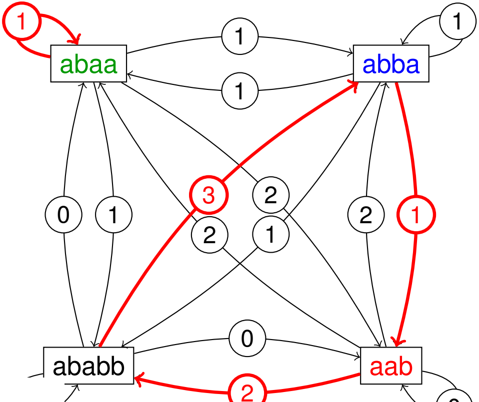




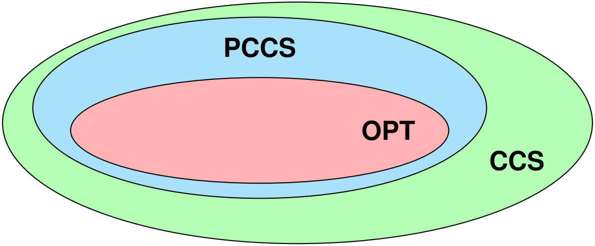
Merging words from a permutation

$$P = \{ababb, aab, abba, abaa\} \quad + \quad \begin{pmatrix} 1 & 2 & 3 & 4 \\ 3 & 1 & 2 & 4 \end{pmatrix} = \sigma_1$$

$$CC(P, \sigma_1) = \quad ab \longrightarrow abb \longrightarrow a \qquad aba$$



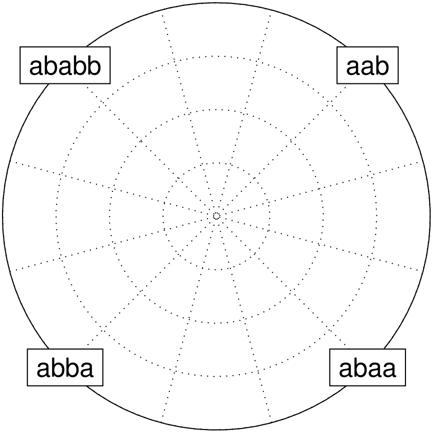
Inclusions of sets of solutions



- **CCS :** Set of Cyclic Cover of Strings.
- **PCCS :** Set of solutions of Cyclic Cover of Strings obtained through a permutation.
- **OPT :** Set of optimal solution of SCCS.

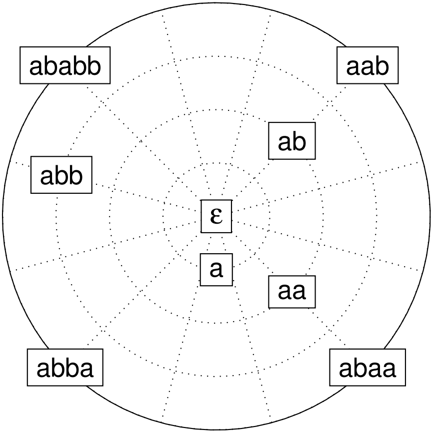
Hierarchical Overlap Graph (HOG)

Hierarchical Overlap Graph

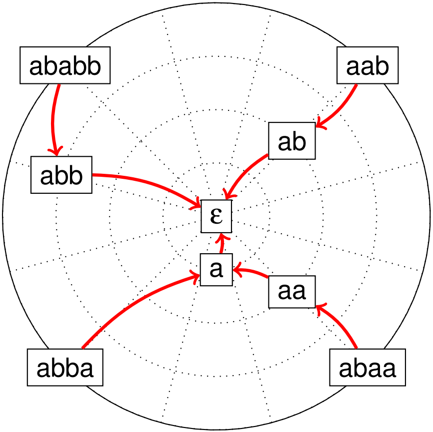


all input words

Hierarchical Overlap Graph



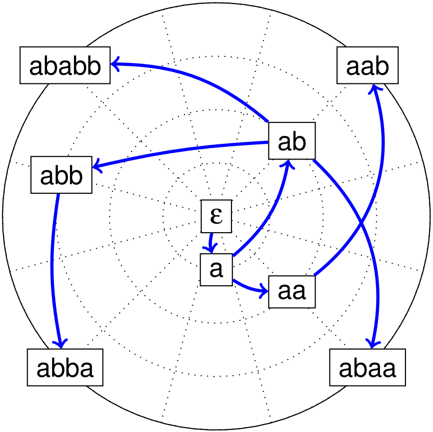
all input words and their maximal overlaps



all input words and their maximal overlaps

red arcs: link a string to its longest suffix

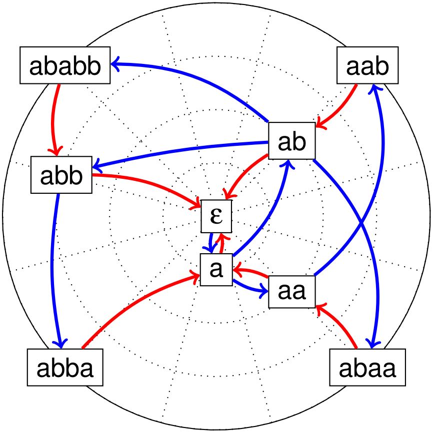
Hierarchical Overlap Graph



all input words and their maximal overlaps

blue arcs: link a longest prefix to its string

Hierarchical Overlap Graph



all input words and their maximal overlaps

A red & blue “path” represents the merge of any two words

Aho-Corasick and `greedy` algorithm for SLS

Aho Corasick automaton

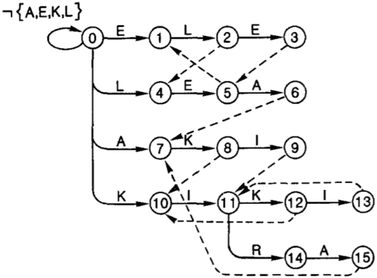
- Part of the 1st solution to Set Pattern Matching [Aho Corasick 1975]
- Search all occurrences of a set P of words in a text T
  1. store the words in a tree whose arcs are labeled with an alphabet symbol
  2. compute the `Failure Links`
  3. scan T using the automaton
- Takes $O(\|P\|)$ time for building the automaton and $O(|T|)$ time for scanning T.
- Generalisation of Morris-Pratt algorithm for single pattern search

Greedy algorithm for SLS [Ukkonen 1990]

Linear time implementation of greedy algorithm for SLS by Ukkonen.

- Simulate greedy algorithm on Aho Corasick automaton of P

- Characterizes states / nodes that are overlaps of pairs of words



$P := \{\textbf{ELE}, \textbf{LEA}, \textbf{AKI}, \textbf{KIKI}, \textbf{KIRA}\}$

Greedy algorithm for SLS [Ukkonen 1990]

Linear time implementation of greedy algorithm for SLS by Ukkonen.

- Simulate greedy algorithm on Aho Corasick automaton of P

- Characterizes states / nodes that are overlaps of pairs of words

LEMMA 3. *Let string u represent state s. For all strings x_j in R, there is an overlap of length k between u and x_j if and only if, for some $h \geq 0$, state $t = f^h(s)$ is such that j is in $L(t)$ and $k = d(t)$.*

Definitions of EHOG and HOG

Extended HOG and HOG

Definition Hierarchical Overlap Graph (HOG)

The HOG of P, denoted by $HOG(P)$, is the digraph (V_H, P_H, S_H) where $V := P \cup \mathcal{O}v(P)$ and P_H is the set:
$\{(x,y) \in (P \cup \mathcal{O}v(P))^2 \mid y$ is the longest proper suffix of $x\}$
S_H is the set:
$\{(x,y) \in (P \cup \mathcal{O}v(P))^2 \mid x$ is the longest proper prefix of $y\}$

Extended HOG and HOG

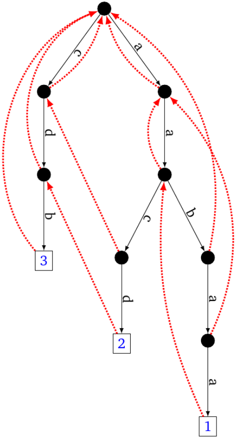
Definition Hierarchical Overlap Graph (HOG)

The HOG of P, denoted by $HOG(P)$, is the digraph (V_H, P_H, S_H) where $V := P \cup \mathcal{O}v(P)$ and P_H is the set:
$\{(x, y) \in (P \cup \mathcal{O}v(P))^2 \mid y$ is the longest proper suffix of $x\}$
S_H is the set:
$\{(x, y) \in (P \cup \mathcal{O}v(P))^2 \mid x$ is the longest proper prefix of $y\}$

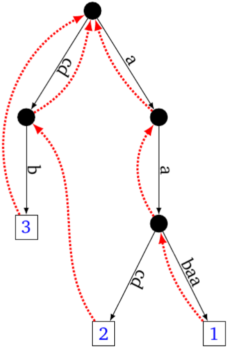
Definition Extended Hierarchical Overlap Graph (EHOG)

The EHOG of P, denoted by $EHOG(P)$, is the directed graph (V_E, P_E, S_E) where $V_E = P \cup \mathcal{O}v^+(P)$ and P_E is the set:
$\{(x, y) \in (P \cup \mathcal{O}v^+(P))^2 \mid y$ is the longest proper suffix of $x\}$
S_E is the set:
$\{(x, y) \in (P \cup \mathcal{O}v^+(P))^2 \mid x$ is the longest proper prefix of $y\}$

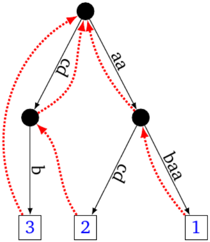
Visual example of construction steps



Aho Corasik tree of *P*



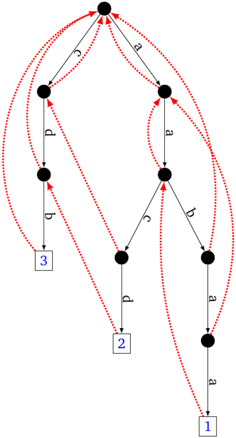
Extended HOG of *P*



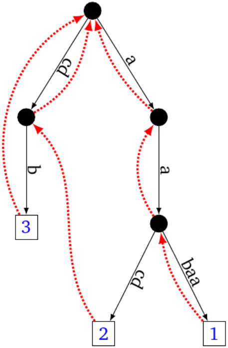
HOG of *P*

Here $P := \{aabaa, aacd, cdb\}$.

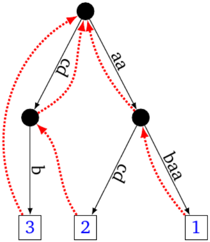
Visual example of construction steps



Aho Corasik tree of P
takes $O(\|P\|)$ time



Extended HOG of P
$O(\|P\|)$ time



HOG of P
time?

Here $P := \{aabaa, aacd, cdb\}$.

Construction algorithm

HOG construction: algorithm overview

Algorithm 3: *HOG* construction

Input: P a substring free set of words; **Output**: $HOG(P)$

Variable: bHog a bit vector of size $\#(EHOG(P))$

build $EHOG(P)$

set all values of bHog to `False`

traverse $EHOG(P)$ to build $R_l(u)$ for each internal node u

run MarkHOG(r) where r is the root of $EHOG(P)$

Contract($EHOG(P)$, bHog)

// Procedure **Contract** traverses $EHOG(P)$ to discard nodes that are not marked in bHog and contract the appropriate arcs

List $R_l(u)$ for a node u of the EHOG

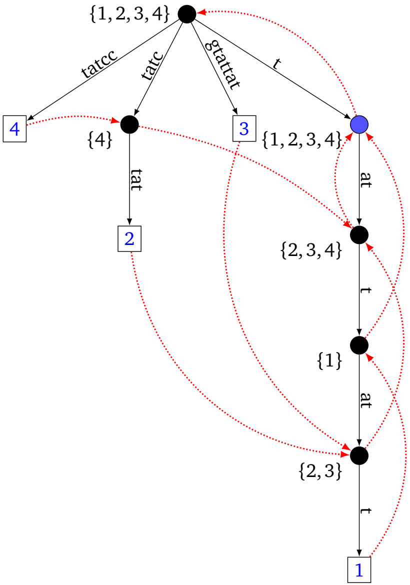
For any internal node u, $R_l(u)$ lists the words of P that admit u as a suffix.
Formally:

$$R_l(u) := \{i \in \{1, \ldots, \#(P)\} : u \text{ is suffix of } s_i\}.$$

- A traversal of $EHOG(P)$ allows to build a list $R_l(u)$ for each internal node u see [Ukkonen, 1990].
- The cumulated sizes of all R_l is linear in $\|P\|$

  indeed, internal nodes represent different prefixes of words of P and have thus different begin/end positions in those words.

EHOG for instance
$P :=$
$\{tattatt, ctattat, gtattat, cctat\}$.



MarkHOG(u) algorithm

Input: u a node of $EHOG(P)$; **Output**: C: a boolean array of size $\#(P)$;

```
if u is a leaf then
    set all values of C to False;
    bHog[u] := True;
    return C;
// Cumulate the information for all children of u
C := MarkHOG(v) where v is the first child of u;
foreach v among the other children of u do
    C := C ∧ MarkHOG(v);
// Process overlaps arising at node u:  Traverse R_l(u)
for node x in the list R_l(u) do
    if C[x] = False then
        bHog[u] := True
    C[x] := True;
return C
```

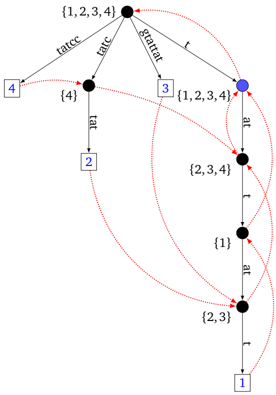
Two invariants

Invariant #1 (after line **7**):
$C[w]$ is `True` iff for any leaf l in the subtree of u the pair $ov(w,l) > |u|$.

Invariant #2 (after line **11**):
$C[w]$ is `True` iff for any leaf l in the subtree of u the pair $ov(w,l) \geq |u|$.

Example for MarkHOG(root)

EHOG for $P := \{tattatt, ctattat, gtattat, cctat\}$.



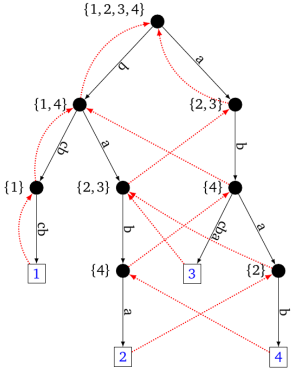
Trace of MarkHOG(root).

node	R_ℓ	C(before)	C(after)	Spec pairs	bHog
ctat	{4}	0000	0001	(4,2)	1
tattat	{2,3}	0000	0110	(2,1) (3,1)	1
tatt	{1}	0110	1110	(1,1)	1
tat	{2,3,4}	1110	1111	(4,1)	1
t	{1,2,3,4}	1111	1111	empty	0
root	{1,2,3,4}	0000 ˆ 0001	0000		
root	{1,2,3,4}	0000 ˆ 0000	0000	(2/3,2)	
root	{1,2,3,4}	0000 ˆ 1111	0000	(1/2/3/4,4)	
root	{1,2,3,4}	0000	1111	(2/3/4,3)	1

Another example

$P := \{abcba, baba, abab, bcbcb\}$

EHOG & HOG



Trace of MarkHOG(root).

node	R_ℓ	C(before)	C(after)	Specific pairs	b
bcb	{1}	0000	1000	(1,1)	
bab	{4}	0000	0001	(4,2)	
ba	{2,3}	0001	0111	(2,2) (3,2)	
b	{1, 4}	1000 ˆ 0111			
b	{1, 4}	0000	1001	(4,1) (1,2)	
aba	{2}	0000	0100	(2,4)	
ab	{4}	0000 ˆ 0100			
ab	{4}	0000	0001	(4,3) (4,4)	
a	{2,3}	0001	0111	(2,3) (3,3) (3,4)	
root	{1,2,3,4}	1001 ˆ 0111			
root	{1,2,3,4}	0001	1111	(1,3) (1,4) (2,1) (3,1)	

Theorem 4

Let P be a set of words.

Then Algorithm 3 computes $HOG(P)$ using

$O(\|P\| + \#(P)^2)$ time and

$O(\|P\| + \#(P) \times \min(\#(P), \max\{|s| : s \in P\})$ space.

If all words of P have the same length, then the space complexity is $O(\|P\|)$.

Can one improve on this?

Conclusion

Conclusions and pointers

- The Hierarchical Overlap Graph (HOG) is a compact alternative to the Overlap Graph (OG).

- For constructing the HOG, Algorithm 1 takes $O(\|P\|)$ space and $O(\|P\| + \#(P)^2)$ time.

  Can one compute the HOG in a time linear in $\|P\| + \#(P)$?

- HOG useful for variants of SLS: for a cyclic cover, with Multiplicities, etc.

Superstrings with multiplicities
Annual Symp. on Combinatorial Pattern Matching, **CPM** 2018
LIPIcs, vol. 105, n. 21, doi: 10.4230/LIPIcs.CPM.2018.21, 2018

More on Hierarchical Overlap Graph. arXiv:1802.04632 2018

In practice

- EHOG as an overlap index

arXiv:1707.05613v1

- A greedy like approximation algorithm for SLS using the EHOG

Practical lower and upper bounds for the Shortest Linear Superstring
B. Cazaux, S. Juhel, E. Rivals
International Symposium on Experimental Algorithms (SEA 2018)
LIPIcs, vol. 103, n. 18, doi: 10.4230/LIPIcs.SEA.2018.18, 2018

Relation to data structures and to assembly algorithms

- Algorithms to compute and update de Bruijn graphs from a suffix tree or a suffix array
  [Cazaux et al, J. of Computer and System Sciences, 2016]
  doi:10.1016/j.jcss.2016.06.008

- How does assembly on a HOG compare to multi-DBG assemblers like SPAdes?
  [Cazaux et al, in Algorithmic Aspects in Information and Management, LNCS vol. 9778, 39–52, 2016]
  Authors version link

Open questions

- How different are EHOG and HOG in practice?

  There exist instances such that in the limit
  the ratio between their number of nodes can goes to ∞
  when $\|P\|$ tends to ∞ with a bounded number of words.
  http://www.lirmm.fr/˜rivals/res/superstring/hog-art-appendix.pdf

- Reverse engineering of HOG

  Recognition of OG by [Gevezes & Pitsoulis 2014]

Funding and acknowledgements

Work on compact EHOG implementation with R. Canovas

Institut de biologie computationnelle

Thank you for your attention!

Questions?