Hierarchical Overlap Graph

B. Cazaux and E. Rivals

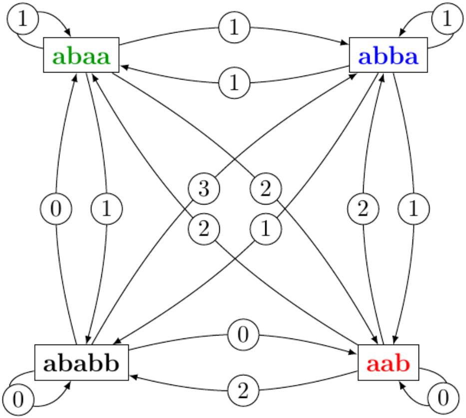
* LIRMM & IBC, Montpellier

8. Feb. 2018

arXiv:1802.04632 2018

Overlap Graph for a set of words

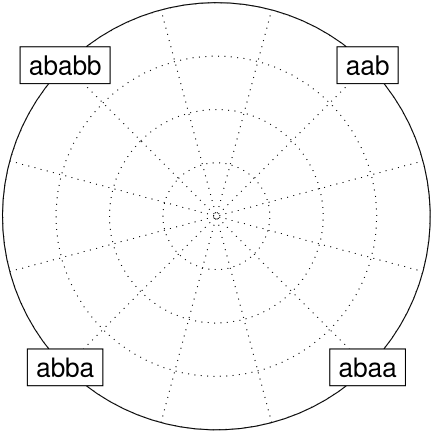
Consider the set
$P :=$
{*abaa*, *abba*, *ababb*, *aab*}



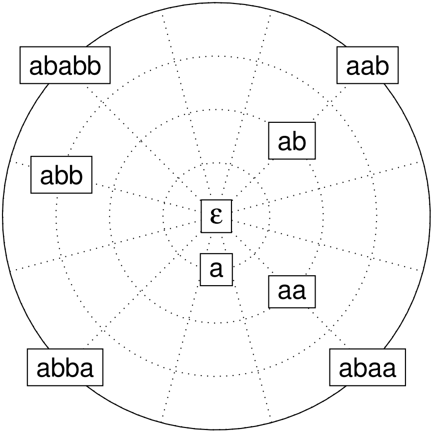
The Overlap Graph (OG) is applied in shortest superstring problems, DNA assembly, and other applications [Gevezes, Pitsoulis, 2011]

- 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 algorithm 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 Overlap Graph
and an algorithm to build it

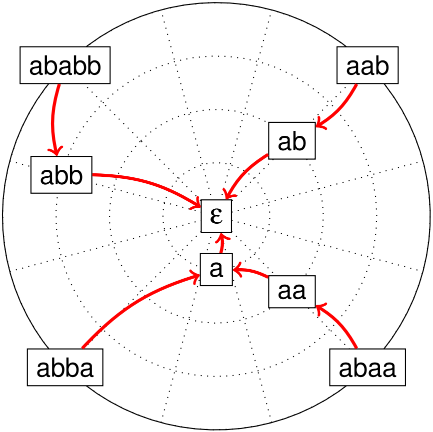


all input words



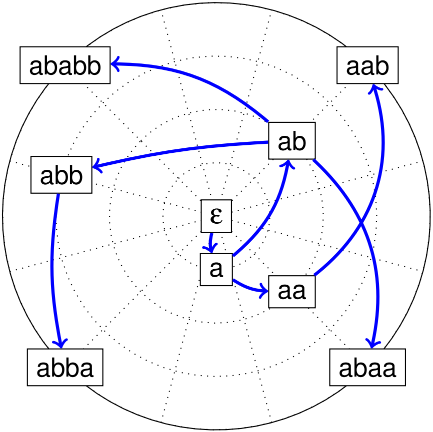
all input words and their maximal overlaps

Hierarchical Overlap Graph



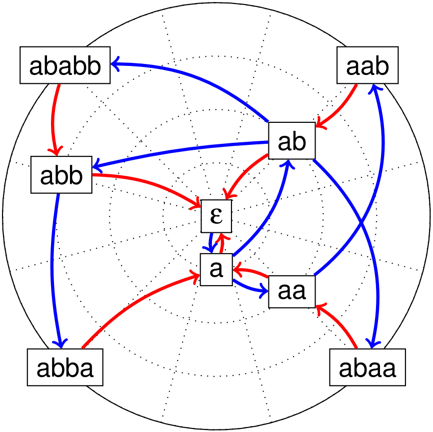
all input words and their maximal overlaps

red arcs: link a string to its longest suffix



all input words and their maximal overlaps

blue arcs: link a longest prefix to its string



all input words and their maximal overlaps

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

Basic definitions

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

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

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

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.

w a b a b b $\boxed{\text{a b a a a}}$

v $\boxed{\text{a b a a a}}$ 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*.

w a b a b b a b a a a

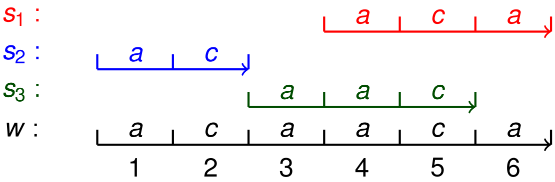
v a b a a a b b b b

u $\underbrace{\text{a b a a a}}_{ov(w,v)}$

Superstring

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.

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]

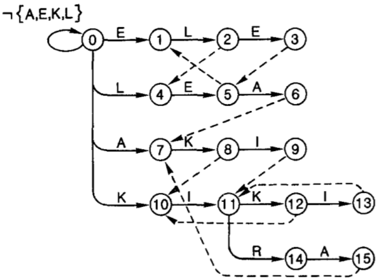
Aho-Corasick and `greedy` algorithm for SLS

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

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



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

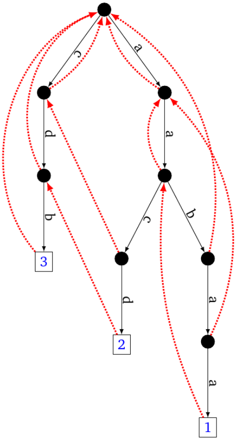
Definition Extended Hierarchical Overlap Graph (EHOG)

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:
$\{(x,y) \in (P \cup \mathcal{O}v^+(P))^2 \mid y \text{ is the longest proper suffix of } x\}$
S_E is the set:
$\{(x,y) \in (P \cup \mathcal{O}v^+(P))^2 \mid x \text{ is the longest proper prefix of } y\}$

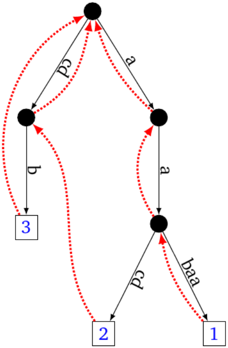
Definition Hierarchical Overlap Graph (HOG)

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:
$\{(x,y) \in (P \cup \mathcal{O}v(P))^2 \mid y \text{ is the longest proper suffix of } x\}$
S_H is the set:
$\{(x,y) \in (P \cup \mathcal{O}v(P))^2 \mid x \text{ is the longest proper prefix of } y\}$

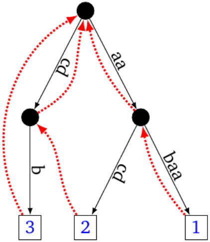
Visual example of construction steps



Aho Corasik tree of P



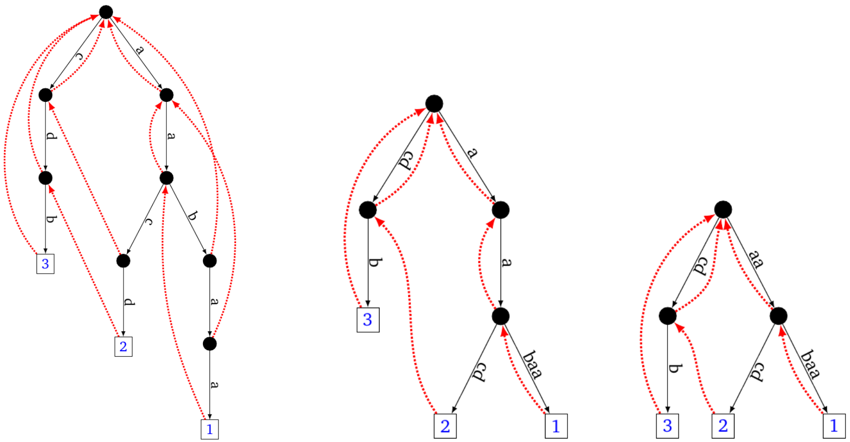
Extended HOG of P



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 1: *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$(EHOG(P), \text{bHog})$

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

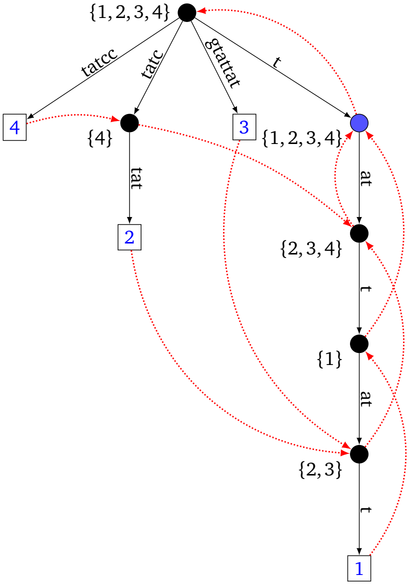
List $R_l(u)$ for a node u of the EHOG

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**

  $C := C \wedge$ 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`

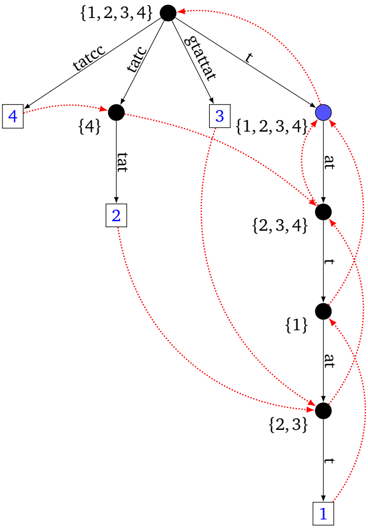
return C

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 $ov(w,l) \geq |u|$.

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



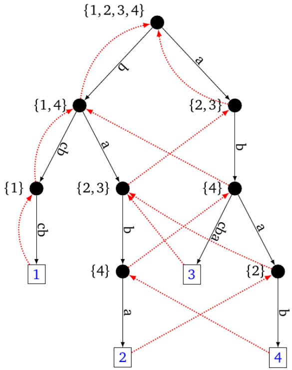
Trace of MarkHOG(root).

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	bHog
bcb	{1}	0000	1000	(1,1)	1
bab	{4}	0000	0001	(4,2)	1
ba	{2,3}	0001	0111	(2,2) (3,2)	1
b	{1, 4}	1000 ˆ 0111			
b	{1, 4}	0000	1001	(4,1) (1,2)	1
aba	{2}	0000	0100	(2,4)	1
ab	{4}	0000 ˆ 0100			
ab	{4}	0000	0001	(4,3) (4,4)	1
a	{2,3}	0001	0111	(2,3) (3,3) (3,4)	1
root	{1,2,3,4}	1001 ˆ 0111			
root	{1,2,3,4}	0001	1111	(1,3) (1,4) (2,1) (3,1)	1

Theorem 1

Let P be a set of words. Then Algorithm 1 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 we improve on this?

Conclusion

Conclusions

- 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.

More on Hierarchical Overlap Graph. arXiv:1802.04632 2018

- Mapping from EHOG to HOG is not a bijection

- 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]

Work on compact EHOG implementation with R. Canovas

Thank you for your attention!

Questions?