Weiner’s Repetition Finder
(with simplifications suggested by the referee)

Vaughan Pratt

Computer Science Department
Stanford University

Combinatorial Pattern Matching 2013
1 Background

- Weiner's starting point: Knuth's 1970 conjecture
- SWAT'73 paper
- Journal submission 1973
Outline

1. Background
   - Weiner’s starting point: Knuth’s 1970 conjecture
   - SWAT’73 paper
   - Journal submission 1973

2. Referee’s Report
   - Conditional accept
   - Condition: make more understandable
Outline

1. Background
   - Weiner’s starting point: Knuth’s 1970 conjecture
   - SWAT’73 paper
   - Journal submission 1973

2. Referee’s Report
   - Conditional accept
   - Condition: make more understandable

3. Existence proof of possibility
   - Algorithm to find all left-longest repetitions so far
   - Suffix splitting as the only complication
1 Background
   - Weiner’s starting point: Knuth’s 1970 conjecture
     - SWAT’73 paper
     - Journal submission 1973

2 Referee’s Report
   - Conditional accept
   - Condition: make more understandable

3 Existence proof of possibility
   - Algorithm to find all left-longest repetitions so far
   - Suffix splitting as the only complication
Knuth conjectured in 1970 that the longest common substring of two strings could not be found in linear time [KMP].

Weiner set out to find a linear time algorithm for this problem.

Morphed into a data structures paper: bi-trees, prefix trees, and associated algorithms.

These solved a more general problem: build the suffix tree of a string in linear time, along with other applications.
1 Background
- Weiner’s starting point: Knuth’s 1970 conjecture
- SWAT’73 paper
- Journal submission 1973

2 Referee’s Report
- Conditional accept
- Condition: make more understandable

3 Existence proof of possibility
- Algorithm to find all left-longest repetitions so far
- Suffix splitting as the only complication
Paper presented at SWAT’73.

(Switching and Automata Theory, renamed Foundations of Computer Science, FOCS, in 1975.)
Outline

1 Background
- Weiner’s starting point: Knuth’s 1970 conjecture
- SWAT’73 paper
- Journal submission 1973

2 Referee’s Report
- Conditional accept
- Condition: make more understandable

3 Existence proof of possibility
- Algorithm to find all left-longest repetitions so far
- Suffix splitting as the only complication
Referee: VP
1. **Background**
   - Weiner’s starting point: Knuth’s 1970 conjecture
   - SWAT’73 paper
   - Journal submission 1973

2. **Referee’s Report**
   - Conditional accept
   - Condition: make more understandable

3. **Existence proof of possibility**
   - Algorithm to find all left-longest repetitions so far
   - Suffix splitting as the only complication
I attended SWAT’73 and listened to Peter’s talk with great interest. The universal reaction seemed to be that the arguments were very intricate, and the question of correctness arose.

When John Hopcroft asked me to referee the paper for CACM, I had no preconceptions about its correctness.

On the one hand, after two weeks I was unable to find any serious error.

On the other I was also unable to “grok” the method.

Perhaps TOC is not my thing after all, and I should go back to NLP...
Two more weeks and I was able to convince myself that something like Peter’s bi-trees could be used to build tries in linear time.

So even if there were any errors, it was no longer necessary to infer that they must be fatal errors.

But why ask each of the paper’s potential $n$ readers ($n = 10? 100? 1000?$) to duplicate my effort if the author could somehow reduce their burden?

Decision: Conditional accept.
1 Background
- Weiner’s starting point: Knuth’s 1970 conjecture
- SWAT’73 paper
- Journal submission 1973

2 Referee’s Report
- Conditional accept
- Condition: make more understandable

3 Existence proof of possibility
- Algorithm to find all left-longest repetitions so far
- Suffix splitting as the only complication
In its present form, even if the paper did contain errors, fixing them wasn’t going to solve anything, as that would do nothing to clarify the paper.

What was needed was to make it clearer why the paper was correct.

My report offered one way of doing this, not as something the author should follow however but merely as an existence proof that much greater clarity was possible.
1. Background
   - Weiner’s starting point: Knuth’s 1970 conjecture
   - SWAT’73 paper
   - Journal submission 1973

2. Referee’s Report
   - Conditional accept
   - Condition: make more understandable

3. Existence proof of possibility
   - Algorithm to find all left-longest repetitions so far
   - Suffix splitting as the only complication
The string BANANAS has 7 symbols numbered 1 to 7 and 8 between-character positions numbered 0 to 7: 0B1A2N3A4N5A6S7.

At each between-character position \( i \), denote the left-longest repetition so far (i.e. ignoring text yet to come) by \( \lbrack \) at \( i \) and a matching \( \rbrack \) at \( j \leq i \).

0. \( \lbrack \)BANANAS 0..0
1. B\( \lbrack \)ANANAS 1..1
2. BA\( \lbrack \)NANAS 2..2
3. BAN\( \lbrack \)ANAS 3..3
4. BAN[A]NAS 3..4 4 repetitions: \( \epsilon \), A, AN, ANA
5. BAN[AN]AS 3..5  # occurrences: 8, 3, 2, 2
6. BAN[ANA]S 3..6 Left longest repetitions are one character
7. BANANAS[\( \rbrack \)] 7..7 short of being a form of suffix identifier.

"[" advances monotonically in this example.
Basic algorithm

while picture is ...[w]a... do
  if $wa$ is a repetition Move "]
  else Move "[" (and "]" in context "[]")

Theorem

"[" always advances monotonically.

Hence $O(n)$ running time assuming $O(1)$ time steps.
Data structure

Realize the movements of [ and ] as movements along edges of a graph G constructed as we go along.

Vertices denote the left-longest repetitions.

They are created as soon as the repetition is first bracketed.

The initial vertex is $\varepsilon$, being considered a repetition from the very beginning.

Edges:

- Link $w$ to $wa$ via an edge labeled $a$. Notation: $w:a = wa$. These edges support ] movement.
- Link $wa$ to its longest proper suffix $u$ in G. Notation: $S(w) = u$. These edges support [ movement.
Algorithm
Start at ε: []BANANAS
While picture is ...[w]a...
   if wa is a repetition follow the a edge, creating wa if necessary
   else if the dotted (S) edge exists follow it
   else at ε: move [] (i.e. get next character)
Outline

1 Background
   - Weiner's starting point: Knuth's 1970 conjecture
   - SWAT'73 paper
   - Journal submission 1973

2 Referee's Report
   - Conditional accept
   - Condition: make more understandable

3 Existence proof of possibility
   - Algorithm to find all left-longest repetitions so far
   - Suffix splitting as the only complication
Suffix splitting: BANANASNA

BANANAS[]NA
Suffix splitting: Create node \( N \)

\[
\begin{align*}
\text{BANANAS}[], \text{NA} & \\
\text{BANANAS}[N]A & \\
\end{align*}
\]
Suffix splitting: Split AN → ε

BANANAS[NA]
BANANAS[N]A

Diagram with nodes labeled ε, N, A, AN, and ANA, connected by arrows indicating the splitting process.
Suffix splitting: Create node NA

BANANAS[NA]
BANANAS[N]A
BANANAS[NA]
Suffix splitting: Split ANA → A

BANANAS[NA]
BANANAS[N]A
BANANAS[NA]
Suffix splitting: METHOD (outline)

Goal: Split \( x \rightarrow wa \rightarrow u \)
where \( u \) is the longest proper suffix of \( wa \) in \( G \) (always exists) and \( x \) is the word in \( G \) if any such that \( S(x) = u \) before \( wa \) enters \( G \) and \( S(x) = wa \) afterwards (\( x \) need not exist).

**Theorem**

*If such an \( x \) exists it is unique, and is determined by the symbol \( c \) in \( wa = vcu \).*

Hence every suffix link \( x \rightarrow u \) can be equipped with an inverse link \( u \rightarrow x \) determined by \( u \) and the \( c \) such that \( x = v'cu \). This \( c \) can in turn be determined as \( c = A[\text{loc}(wa) - \text{len}(u)] \) since \( S(x) = wa \).

**Method:**
- Find \( u \) using \( w \) and \( a \).
- Find \( x \) using \( u \) and \( c \) as in the theorem.
Suffix splitting: Finding $u$ and $x$

We must find $u = S(wa)$ at the creation of each new node $wa$, whether or not $x$ exists. Do so as follows.

Before linking $w$ to $wa$, set $t = w$ and then repeatedly set $t = S(t)$ (i.e. follow dotted suffix links) until either $ta$ exists or $t = \varepsilon$. Take $u$ to be $ta$ if it exists, else $\varepsilon$.

This is still $O(n)$ because the next $w = S(w)$ skips over all the steps taken by $t = S(w)$ in a single step.

To find $x$ we furnish every suffix edge $x \rightarrow u$ of the graph with its inverse $u \rightarrow x$. Although there may be multiple $x$ satisfying $S(x) = u$, only one can “factor through” $wa$. (Connection with Weiner’s algorithm: these are the edges of a compacted suffix trie.)

To find $x = v'cu$, determine $c$ as $A[loc(wa) − len(u)]$. 
We can now list all 6 fields of a vertex of G.

- \( \text{loc}(w) \) location of 1st occurrence of \( w \)
- \( \text{len}(w) \) length of \( w \)
- \( S(w) \) longest proper suffix of \( w \) (as a vertex in \( G \))

The above three fields are fixed at the time \( w \) is created as \( w = v : b \).
\[ \text{loc}(w) = v . b, \text{len}(w) = \text{len}(v) + 1, \text{and } S(w) = u (t : vb \text{ or } \epsilon). \]

The remaining 3 fields are \( \Sigma \)-indexed sparse arrays.

For each symbol \( a \in \Sigma \):

- \( w.a \) Location of the first occurrence of \( wa \) (right end). Set at the later of creating \( w \) or 1st occurrence of \( wa \) (next slide).
- \( w:a \) Vertex of \( G \) denoting \( wa \).
  Set when \( wa \) is created (see next slide).
- \( *a:w \) Inverse suffix link.
  Set when the corresponding suffix link is created (always paired).
Detecting repetitions

In the context ...$[w]a...$, $wa$ is a repetition when $w.a$ is defined, namely as the location of the first occurrence of $wa$.

$w.a$ is stored at node $w$ in $G$ either at creation of $w$ or later.

- *At creation*: When $w$ is created in $G$, record for each symbol $a$ the location of the first occurrence of $wa$ in node $a$. Notation: $w.a$. To do this, either copy all $x.a$ to $w.a$ when $x$ exists, otherwise set $w.a$ to $\text{loc}(w) + 1$ where $a$ is the letter at that location.

- *Later*: Whenever the repetition test for ...$[w]a...$ fails, set $w.a$ to be the current position in the string.

Main theorem:

**Theorem**

*For all $w \in G$, if $wa$ occurs in the string then $w.a = \text{the location of the first occurrence of } w$.***
To find patterns in a string $A$.

Dumb method (how my referee’s report envisaged doing this):

- Apply the algorithm to $A$.
- For each pattern $P$ continue the algorithm with input $P$, where $\$ \notin \Sigma$. This produces a graph $G_{A\$P}$ as though having processed $A\$P$ in one pass.
- When done with each $P$ restore $G_{A\$P}$ to $G_A$ (routine).
Relation to Weiner’s algorithm

Take VP* to be my variant reversed so both scan right to left.

Essential common feature of PW and VP*: both find prefix identifiers of position.

The edges of PW’s compacted suffix trie (i.e. suffix tree) are VP*’s $a; w$ edges.
Corollaries

Corollary 1: Instead of the above dumb method, VP can do pattern matching without modifying $G$, namely by scanning the patterns right to left and navigating in $G$ via the inverse suffix links $*a : w$ instead of the $w : a$ links.

My report did not make that connection with PW and hence overlooked that possibility.

Corollary 2: VP and PW differ only in implementation details. (This was not clear to me until this morning.)