Mobile device detection based on user agent strings

Problem presented at South African MISG 2011
brought by Zyelabs
Industry: Rumbidzai Mukungunugwa, Ismail Dhorat

Participants: Colin Please, Ludovic Tangpi, Asha Tailor, Dario Fanucchi, Byron Jacobs, Shaun Kimmelman, Graeme Hocking

January 14, 2011

Overview

The Goal

- A “user agent” string is sent from device to a server to identify itself and its characteristics, i.e. device type, screen size, ... e.g.

  Mozilla/5.0 (iPod; U; CPU iPhone OS 3_1_1 like Mac OS X; en-us) AppleWebKit/528.18 (KHTML, like Gecko) Mobile/7C145

  BlackBerry7100i/4.1.0 Profile/MIDP-2.0 Configuration/CLDC-1.1 VendorID/103

- “User Agent Strings are **not** standardized
- The string is compared to a list on `WURFL` (Wireless Universal Resource File) database to get the best possible match.

The problem

- Database has (currently) 13,000 entries and growing
- Strings are in no particular format, length or order
- Often there is no perfect match - user agent strings may be in a different order, use different abbreviations, wrong, etc. ...
- Currently, *Levenshtein* algorithm is used to match strings and the whole database is searched.
- **MISG: Wish to get the best (or satisfactory) match in shortest time**

M	o	z	z	i	l

M	o	z	i	l	l	a	3

The group considered;

- string matching algorithm - literature search
- implementation of existing method (for comparison)
- “quick” improvements
- subdivision of database
- improved database storage algorithms
- future possibilities

Exact String Matching

Involves *alignment* and *matching*

```
S = establish
T = Antidisestablishmentarianism
```

```
Antidisestablishmentarianism
          establish
```

- **Brute Force**
  Easy to implement. Worst case $O(m \times n)$
- **KMP (Knuth-Morris-Pratt)**
  Good scaling, bad hidden constant
- **Boyer-Moore Algorithm**
  Industry Standard text searching algorithm

Inexact String Matching

Try to find the *best fit* for two strings

```
S = fragile
T = Supercalifragilisticexpialidocious
```

```
Supercalifragilisticexpialidocious
          fragile
```

- **Hamming Distance**
  Number of positions at which aligned symbols are different
- **Edit Distance (Levenshtein)**
  Smallest number of *edits* from S to T
- **Longest Common Subsequence**
  Longest subsequence in both strings. Eg. *diff* in Unix
- **Longest Common Substring**
  Longest common substring between the two strings

Fast Dynamic Programming Methods for each of these exists

Existing Brute force method

Brute force: Make a full search, and compare everything, e.g. in python,

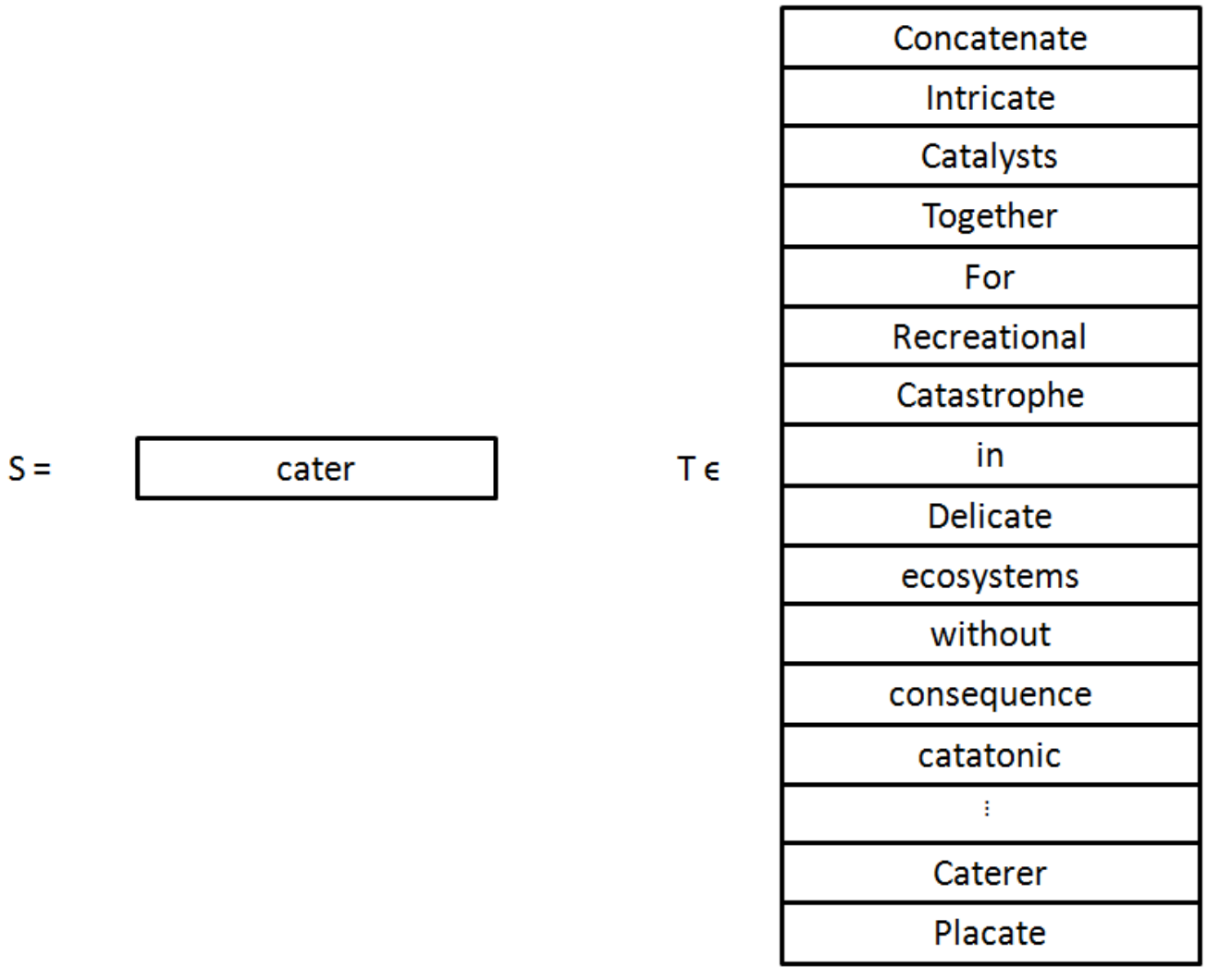
```
for i in xrange(len(database)):
    a = lev2(database[i], user)
    if a <= b:
                b = a
                j = i
return database[j]
```

Running time: **0.316**.

First improvements

- Fix a threshold and stop when happy with the distance.
  - Running time in the "worst" case: **0.419**.
  - Running time in the "best" case: **0.128**
- Subdivide the database per order of priorities.
  e.g. S = ['Nokia', 'Samsung', 'Ludovic','Acer']
  (only search the relevant category)
  - Running time in the "worst" case: **0.286**.
  - Running time in the "best" case: **0.074**.
- Order the database in frequency of request over last week, e.g. stop when happy with distance. (Not able to implement without data)

Matching to a database of strings



Repeated String Matches / "Brute Force"

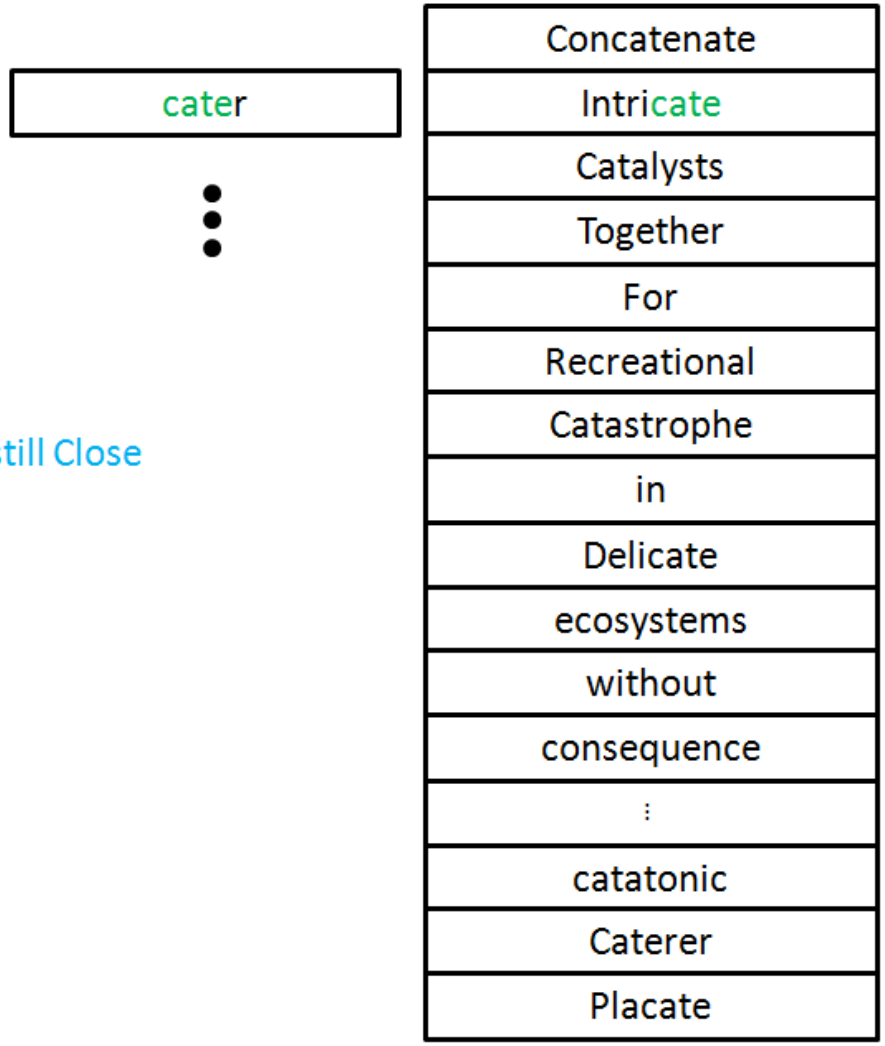
cater

Concatenate
Intricate
Catalysts
Together
For
Recreational
Catastrophe
in
Delicate
ecosystems
without
consequence
⋮
catatonic
Caterer
Placate

Hamming Dist over substring: Close

Levenshtein: Not as close! (many deletes)

Repeated String Matches / "Brute Force"



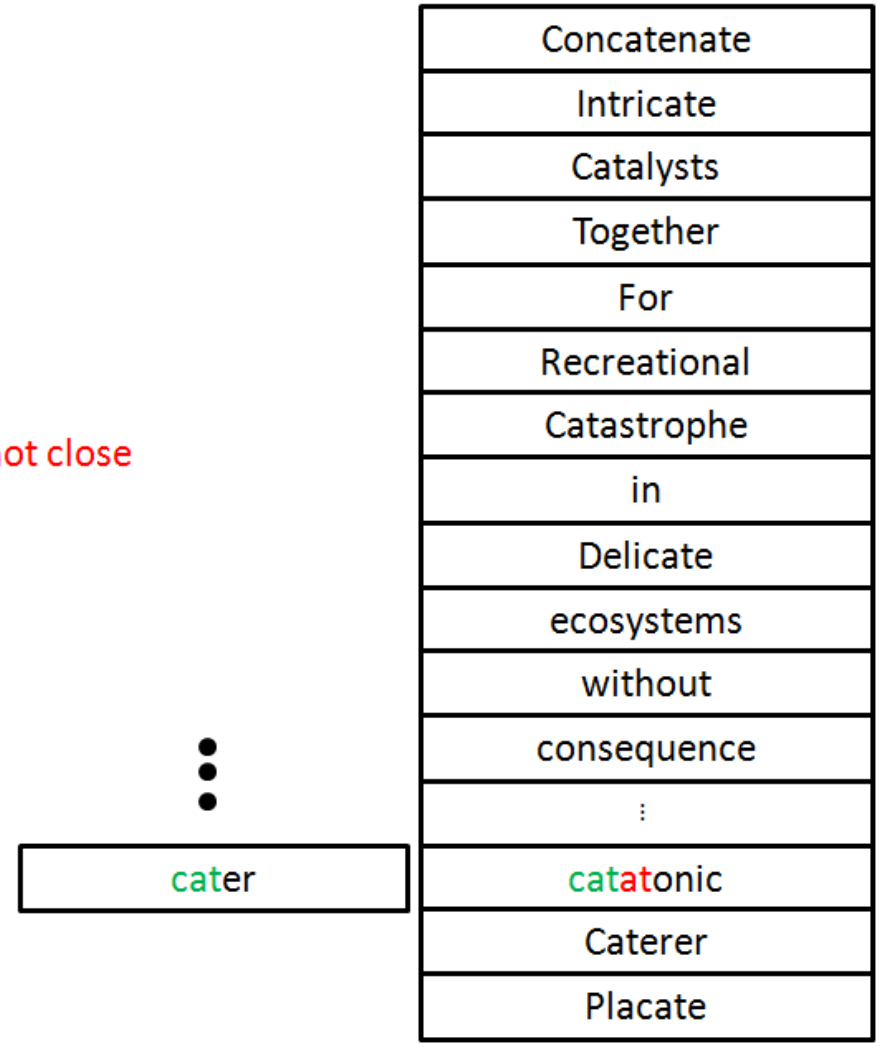
Hamming Dist over substring: still Close

Levenshtein: Still not close

Repeated String Matches / "Brute Force"

Hamming Dist over substring: not close

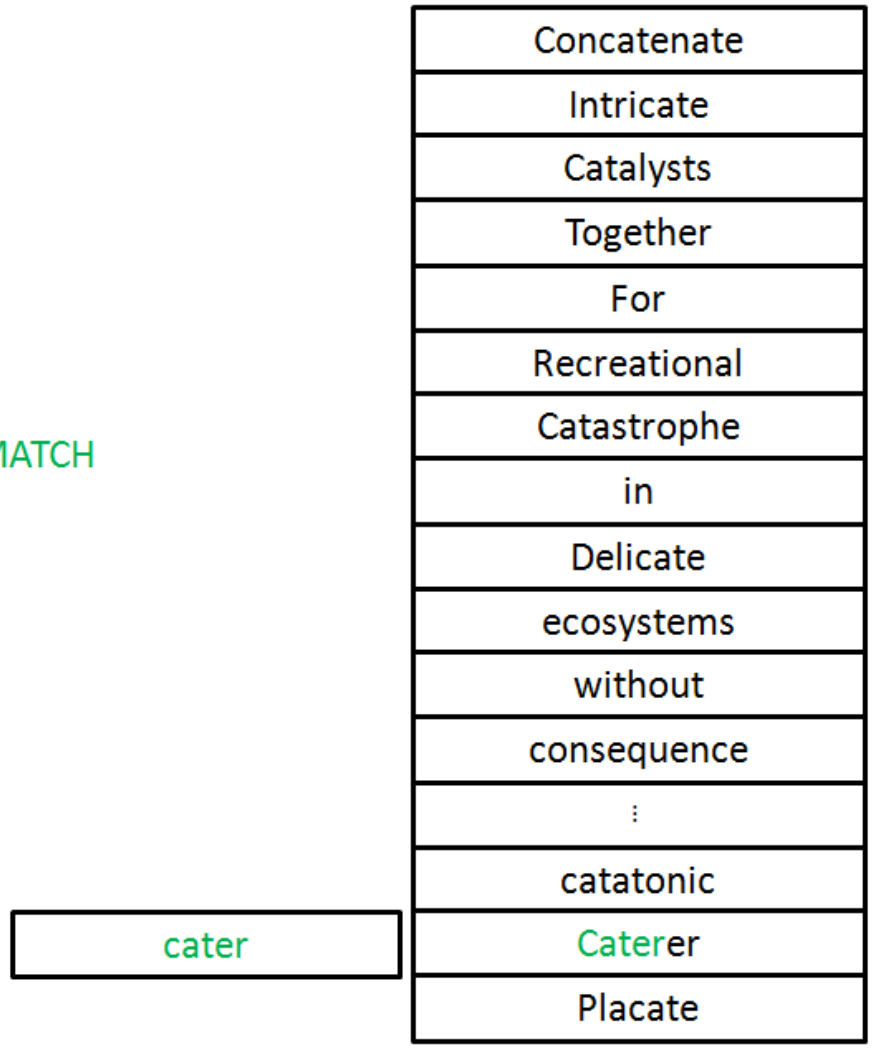
Levenshtein: Not close



Repeated String Matches / "Brute Force"

Hamming Dist over substring: MATCH

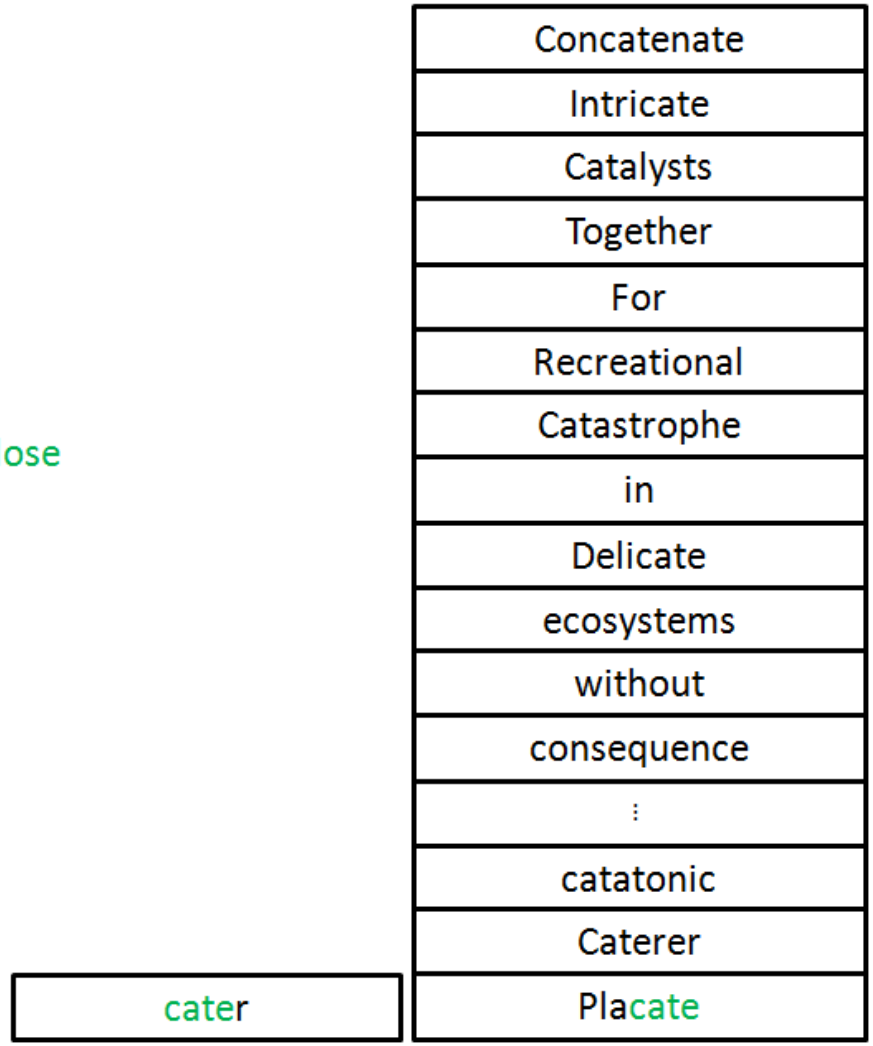
Levenshtein: close



Repeated String Matches / "Brute Force"

Hamming Dist over substring: close

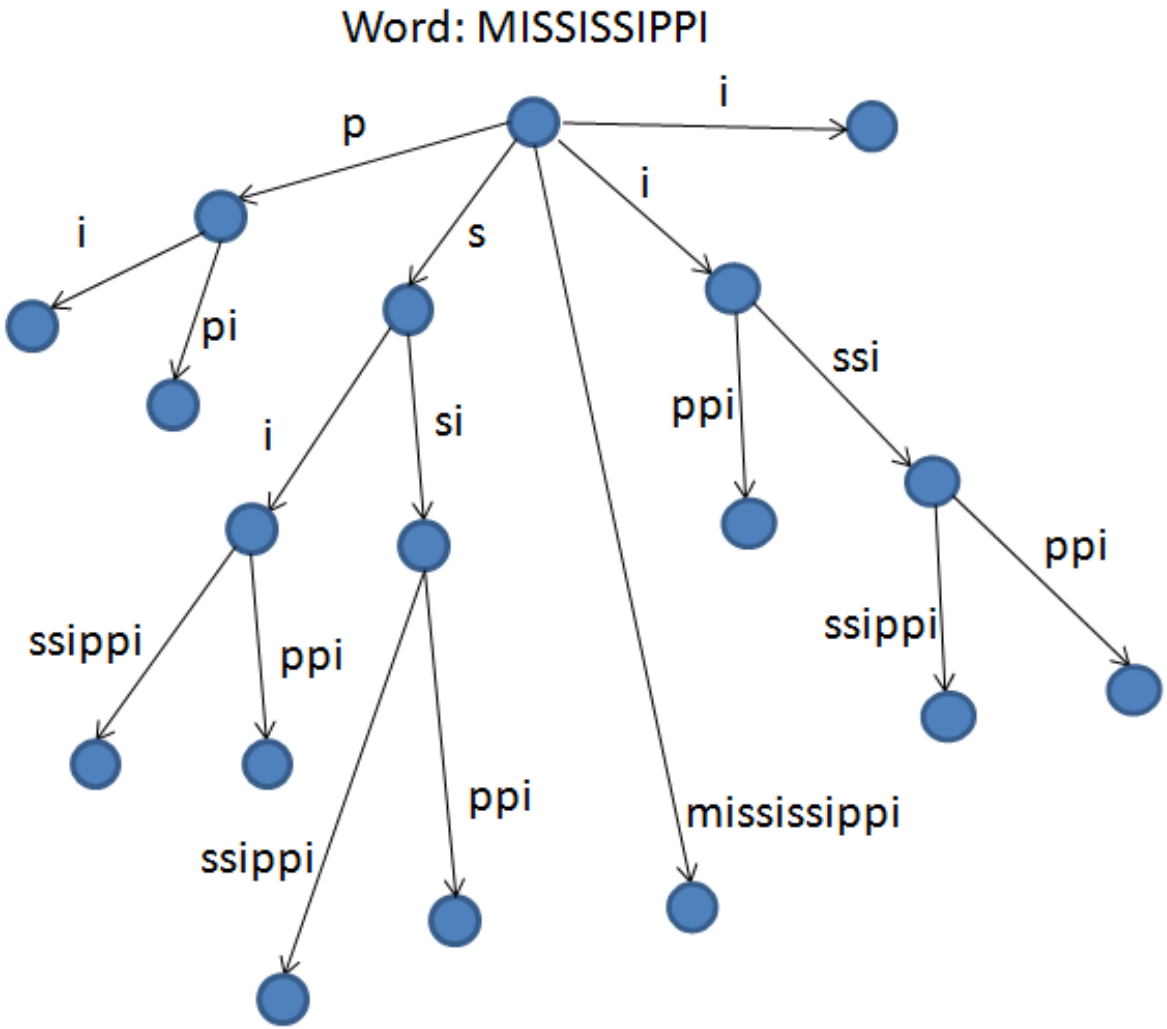
Levenshtein: relatively close



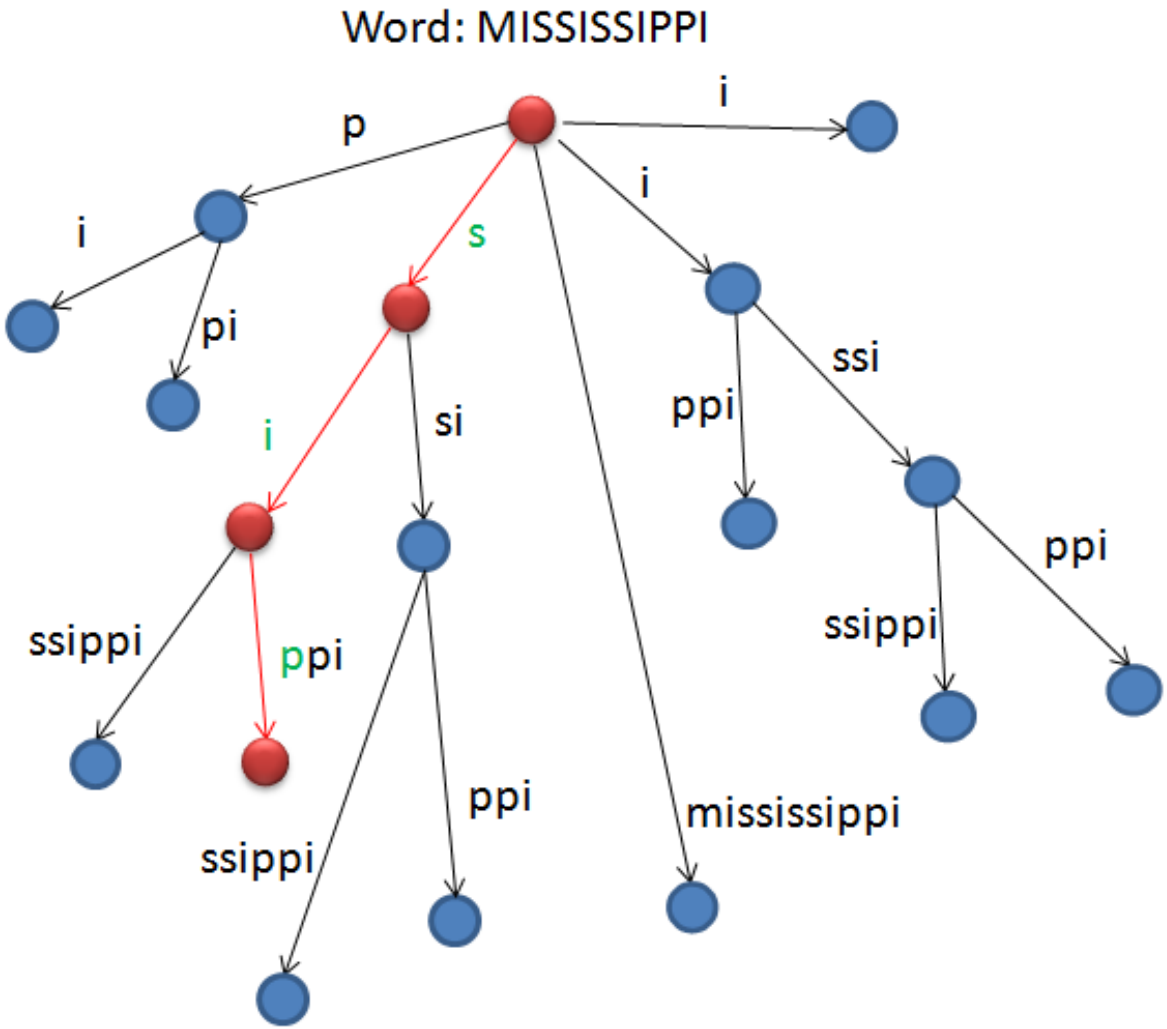
Improvements in the Literature

- Repeated Matching is $O(k \times C(n, m))$, where $C(m, n)$ is the cost of a string-matching algorithm
- We can do **MUCH** better!
- Aho-Corsaik Algorithm for EXACT phrase matching
  - Based on the KMP algorithm
  - Achieves $O(C(m, n) + k)$. This is very good: $k \approx 14000$ and $m, n \approx 200$
  - But EXACT matching leads to high error rate!
- Suffix Trees for phrase matching
  - Pre-processing to construct a *suffix tree* of the database
  - $O(m)$ search times!
  - space concerns and exactness concerns

Suffix Tree Example



Suffix Tree Example

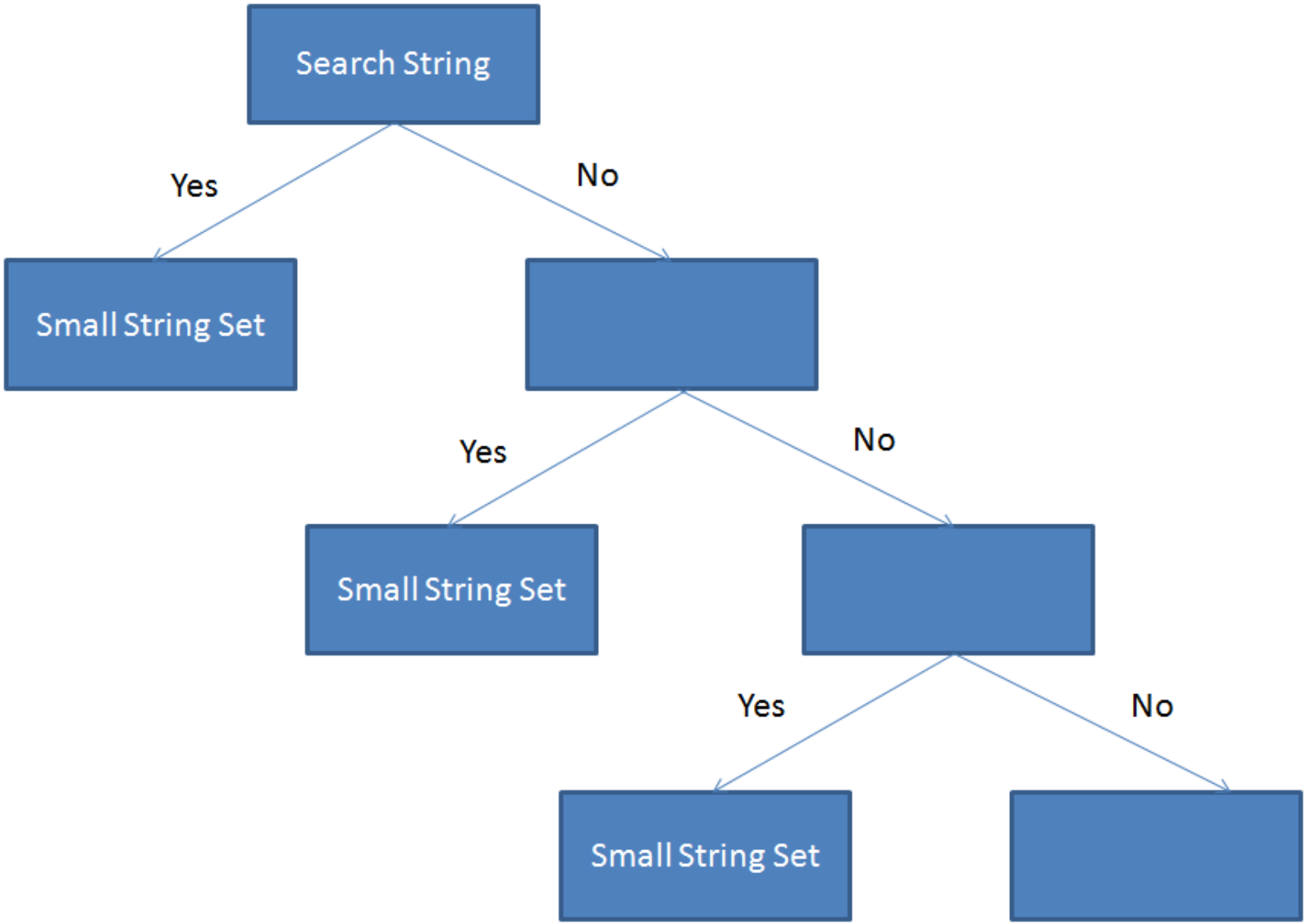


Binary Strucutre

- **Inexact Search**
  We only need a substring to match
- **Time and Speed increase**
  The performance is increased over Brute Force.
- **Preprocessing**
  Our algorithm is general in that the tree is generated based on the database.
- **Inexactness**
  This is characterized by irrelevant data; i.e. We never create sub-group based on irrelevant data
- **Frequent User-Agent Strings**

$$C[1] \gg C\left[\left(\frac{10}{100}\right)\frac{9}{10} + \left(\frac{90}{100}\right)\frac{1}{10}\right] \approx C \cdot 18\%$$

Binary Structure: Visual Aid



Binary Structure: Visual Aid

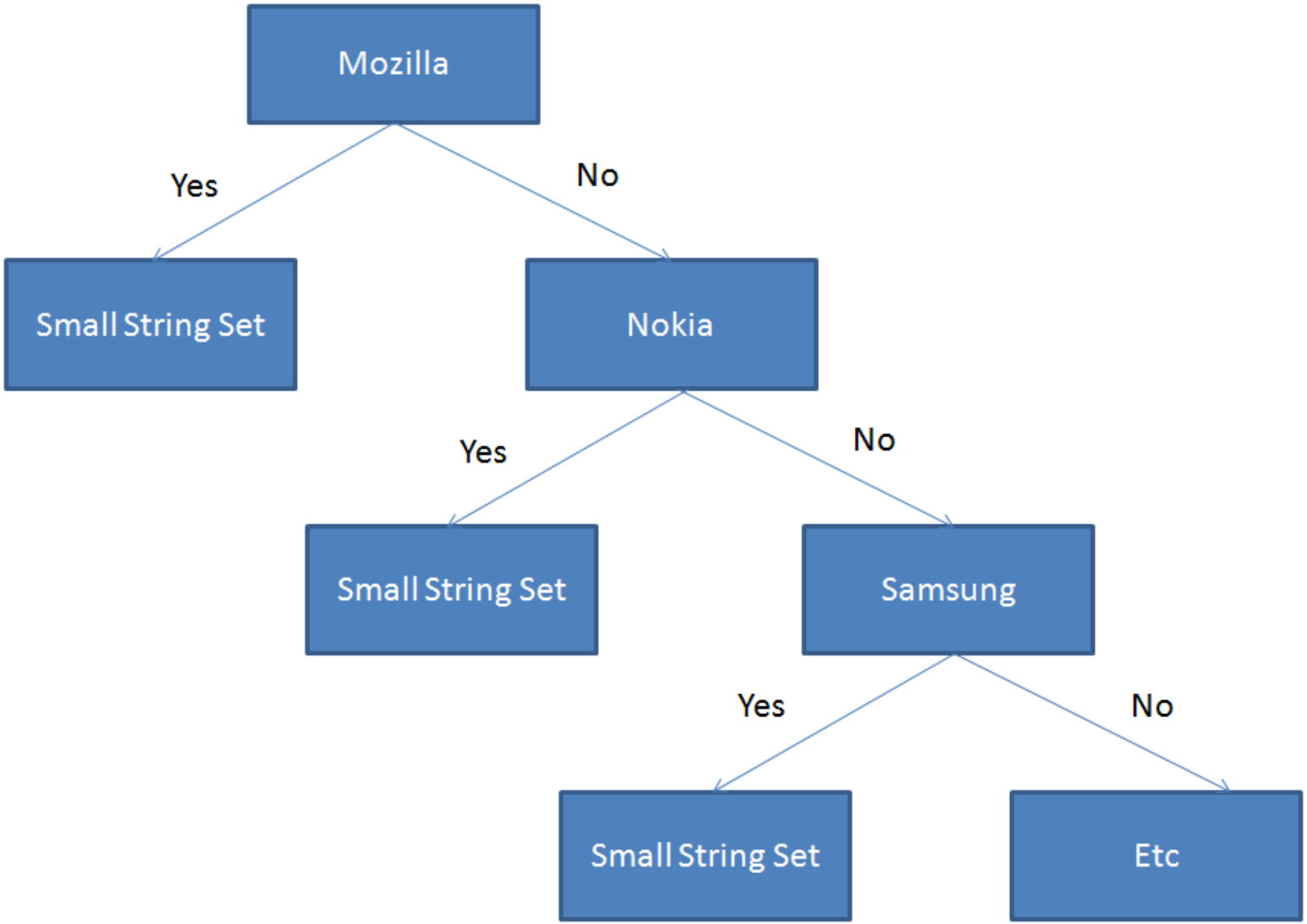


Table of Results

Algorithm	Time Range	Comment
Basic Brute Force	0.316 – 0.419	No Pre-processing
Threshold	0.128 – 0.419	No Pre-processing
Subdivision (simple, threshold)	0.074 – 0.286	Pre-Process
Caching	–	extra coding
Order by Popularity	–	unable - promising
Suffix Trees	–	Huge potential

Final Remarks

- Have understood and programmed the algorithm
- Thorough literature search of string matching & search algorithms
- Implemented one or two schemes (relatively simple) that have given good improvement
- Identified methods that will give substantial improvements (e.g suffix trees in storage) and begun implementing them ....
- **Conclusion** - a range of possible improvements have been suggested, that may be used in combination or separately, all of which will give significant improvements to the method, some spectacular!