PLDI 2024 Paper #132 Reviews and Comments

==========

Paper #132 PTV: Better Version Detection on JavaScript Library Based on Unique Subtree Mining

Review #132A

==========

Overall merit
---
2. Weak reject

Reviewer expertise
---
2. Some familiarity

Paper summary

This paper presents an approach for detecting versions in JavaScript libraries. The idea is to represent different versions as trees and then at runtime to check which version applies to the current library. As this is expensive the authors propose some techniques to reduce these trees by eliminating the common parts in different trees and leveraging algorithms such as PathColoring and MinCoverSet. The approach has been implemented and compared with another web detector called LDC.

Comments for authors

The paper is clearly written although it has some small mistakes (please see below). The approach is based on another recent paper [PTDetector 2023] which already uses tree data structures to extract various features from libraries. The approach is limited in the sense that it can only detect versions that it has already seen; it seems tedious to adapt for new versions. Perhaps the authors can sketch some incremental procedure to update the "forest" when new versions are released. Further, it is unclear if the addressed problem is significant as it seems a fix would be to add proper versioning to each library.

Small comments:
* I wonder what the difference is between "minified" and "minimized"
* Line 128 what is "commonness" here? How common is it to have conflicts?
* Line 136 remove "requires" or "is".
* Line 165: "can we concluded" -> "can we conclude"
* Line 170: *to* "identify
* Line 187: *the* "most unique structure"
* Line 323: the algorithm is described only informally.
* Line 557: "full path" -> "full paths"

Review #132B

==========

Overall merit
---
3. Weak accept

Reviewer expertise
---
2. Some familiarity

Paper summary

The paper proposes a new technique for detecting the library versions used by websites. The paper motivates the need for such detection as means to identify vulnerabilities introduced by defective library usage, website profiling and sales engineering. The paper proposes PTV that builds on PTdetector, for matching the library accessed by the website to the correct version of the library, given the set of library versions. The main idea is to construct pTree (a tree representation of JavaScript code) for the accessed library and each of the given versions. Each version group representation is minified, so there is no overlap between the versions. This processing is done offline. Then during runtime, the accessed version of the library will be matched to exactly one of these trees (which can represent multiple versions), as only one of them can be extended to the pTree of the accessed version. The paper proposes an algorithm for constructing the minified pTrees of the versions and a simple algorithm for matching at run time. The paper evaluates the effectiveness of this technique on 64 CDNJS libraries and compares it against LDC.

Comments for authors

I believe the overall motivation of the paper; it is important to identify library versions used by webpages precisely. The solution proposed by the paper seems novel, unfortunately I couldn't completely comprehend it despite multiple reads due to presentation issues. The evaluation seems somewhat convincing, but I find it lacks clarity and have some concerns.

According to me, one of the main issue with this paper is the lack of clarity in presentation and annoying typos. The paper reads well in the first two sections, but it completely deteriorates in section 3 onwards. For instance, the formal descriptions of various lemmas should clue in the reader, how the solution maybe forming. Unfortunately, it felt like random definitions that just popped up without enough context. Also, the sheer number of symbols defined was exhausting and I had to do multiple back and forth to look up what was being defined. Perhaps using descriptive symbols such as paths(T), fullPaths(T) would make it easier on the reader than using T.P and T.P_f. Consider using running examples to explain the working of all algorithms. While there is some use of examples, it wasn't sufficient to understand the technique.

What is the overall workflow you envision for using PTV in browsers? Do you expect the versions computations to be run each time the page loads or do you think it should be cached and updated frequently? If so, how frequently does it need to be run. Does the time overhead for version detection affect the browsing capabilities? How is the version stubs expected to be stored in the browser and can you determine what libraries need to be versioned?

Is CDNJS the largest provider of libraries? If not, what is the max number of libraries are hosted on other providers and do you anticipate it to scale well. Also, what do you expect to happens when one uses more than one CDNs. Please comment.

I could not follow how to interpret the data presented in Figure 8 and 9. The data that we need to see is how frequently LDC is wrong. When it is right, how many times the precision offered by PTV was higher/lower than that offered by LDC. What is the precision difference in best case, average case and worst case?

Nits:
Line 135: Garbled sentence.

Line 202: Use passive tone.

Line 224: ... we are easy to know that... → it is easy to see that

Line 236: Missing punctuation.

Line 236: ... though adding root and .... → through adding root and... ?

Line 260: of T with respect to ... → of T contained in ?

Line 276: What variable are you referring to here?

Line 277: Use set notation to make it consistent with the rest of the paper for B^*_ϕ and $\neg$$B^*_\phi$.

Corollary 3.3.1: It seems this should be true even if S1 is not a subset of S2.

Lemma 3.3.1: Please also add the formal description here. Hard to follow with textual description.

Line 325: What algorithm are you referring to here? There was no algorithm in previous section.

The line 3 in Algorithm 2 needs to be more formal. The text is hard to parse.

Line 422: Incorrect description of set $\bar{\Omega}$

Review #132C
==========

Overall merit
---
1. Reject

Reviewer expertise
---
3. Knowledgeable

Paper summary

This paper presents the design, implementation, and evaluation of PTV, which is a technique to minify trees used in library detection.
The new technique is implemented on top of PTDetector, which is a tool to detect which libraries, and which versions, a particular JavaScript webpage is using; previously published at ASE [Liu and Ziarek 2023].
The main idea of the technique is to extract the most unique structure of each pTree, and only save that instead of the whole tree.
The paper evaluates PTV against 64 libraries, comparing the storage saved by PTV's miniaturization to PTDetector, the soundness of PTV by manually inspecting each detected library version (not compared against PTDetector), and the time required to minify pTrees.
The paper also evaluates PTV's soundness, and accuracy against Library-Detector-for-Chrome (LDC).

Comments for authors

Pros:
+ The paper compares the soundness of PTV against LDC
+ The paper describes the tree miniaturization in detail

Cons:
- The evaluation does not evaluate the soundness of PTV compared against the base technique PTDetector
- The evaluation does not compare the time required for library detection for each of the three techniques: LDC, PTDetector, and PTV
- The paper does not evaluate independently the ability to detect a library vs the ability to detect the correct version of that library
- The evaluation does not list how many version of each of the 64 libraries were used for the evaluation
- Unsure how the technique scales to more libraries

The main contribution of this paper is the novel algorithm to minimize the size of pTrees using in previous work.
Unfortunately, the paper does not evaluate the benefit and impact of the tree miniaturization process to an acceptable level.
The paper claims that the soundness of PTV and PTDetector is the same.
I appreciate that the description of the algorithm explains the relationship between the miniaturized tree and the original.
However, this claim needs to be validated experimentally.
The paper does not evaluate the soundness of the original technique, PTDetector, versus the new technique, PTV.

Besides comparing the soundness, the paper does not compare the time required to perform library detection with the competing techniques LDC and PTDetector.
Section 5.5 simply breaks down the time required for PTV to minimize trees, but does not offer any insight into the performance of library detection using those minimized trees, and compare it against LDC and PTDetector.

The evaluation also conflates different libraries with their different versions.
There are two separate research questions here: (1) Can PTV accurately detect which libraries a particular website is using, and (2) can PTV accurately detect which version of those libraries are being used?
The evaluation mixes both questions.

Separating libraries from versions is especially pertinent when the corpus consists of 3,664 versions of only 64 libraries.
This is an average of around 57 versions per library.
Is this how the data is distributed?
Or is it the case that a few libraries have a very large number of versions, which most having few versions?
Also, what is the size of the pTrees per library and per version, in average?

It appears to scale better than LDC, which requires hand-written tests per each library.
However, L22 states that there Cndjs now contains 6,056 libraries, and the paper only uses 64.
Can PTV scale to all the libraries in Cndjs?

Review #132D
==========

Overall merit
---
2. Weak reject

Reviewer expertise
---
2. Some familiarity

Paper summary

The paper discusses version detection of JS library in web pages using unique subtree mining. While existing work help detect libraries, this work is positioned as being able to detect versions of libraries due to their implication on “sales intelligence”, website profiling, etc. The idea at its core is to determine a unique subtree among a set of trees and using that to minify the trees. The paper formalizes this idea by defining various properties and provides an implementation overview. The experimental section discusses four research questions: effectiveness of minification, soundness, accuracy and time overhead. Unfortunately, I was unable to interpret the results.

Comments for authors

The paper is very well written up to Section 3.2 and lot of thought seems to have gone into the design of the algorithms. However, the presentation style overwhelm the reader with details making it difficult to follow and makes it easy to miss the forest for the trees (pun intended). Even after multiple reads, the approach was unclear to me. While the details help with academic rigor, the lack of a high level exposition of the concepts and how they come together to solve the problem at hand makes the paper difficult to follow. I recommend that the authors consider adding a Overview section after Motivation discussing the entire approach using illustrative examples before formalizing the discussion. Surprisingly, the experimental section also does not provide crisp details on the setup and the comparison between approaches. The long winded discussion makes it difficult to interpret the results too.

Beyond presentation, there are other concerns. On the one hand, the motivation for this work discusses the use of such version detection for “sales intelligence”, security analysis, among other applications. What does sales intelligence mean? If this is useful for security analysis, how are the results from this approach being used as part of the security analysis? What kind of vulnerabilities were detected? Without a clearly articulated client application that uses this setup, the approach seems to be operating independently and it is hard to appreciate the technical contributions.

The experimental comparison is performed with LDC which can recognize versions for only 83 libraries, when the overall domain of libraries is more than > 6K. If the detection rate is so low, isn’t LDC a strawman? And if the problem was so compelling, wouldn’t there be other tools in this space (given LDC has been in development since 2010)? In

the experimental section, the corpus of libraries used for version detection is reduced to 64 libraries. So, while the early parts of the paper talks about scalability concerns with the number of libraries and versions, the experiments seem to have been performed on a small fraction.

Given these concerns, I am not convinced the paper is ready to be accepted at PLDI.

More comments:

* Line 25: “used for competitor analysis, sales intelligence, security analysis, and website profiling.” What does this mean? How are they used for competitor analysis or sales intelligence?

* Line 62: “ outperforms the existing methods on all” will be good to mention by how much. What is the SoTA, and what is PTV’s accuracy?

* Line 129: “the commonness of global property conflicts in JavaScript” what does this mean?

* Line 165: concluded → conclude

* Line 170: algorithm “to” identify
* Line 200: “Careful readers will notice that we are not able to find a unique subtree for the pTree of version A. ” — I would like to think of myself as a careful reader but don’t follow why there cannot be a unique subtree, given versions B and C.

* Line 202: “You will understand why it is safe to do so in Sec. 3.4.” please avoid forward reference without providing some details.
Section 3.2 is very well written.

* Line 263: “Easy to see ...” — Rephrase into a full sentence

* Line 279: Discussion from hereon to the end of the subsection is quite dense. Will be good to expand and provide illustrative examples to help readability

* Line 422: “Then we can get the value of S(T1) by union all the ω.” How? Wouldn’t union include all trees? Did you mean intersecting?

* Line 422: Is the inverse coloring collection accurate? If so, the details in the Table for this is inaccurate

* Section 3.5 onwards till the end of the section is hard to parse. Will be good to make it more reader friendly. Explain the high-level idea on how the approach works before diving into the details.

* Line 964: “Compared to widely-used machine-learning-based detection methods”? What are these ML methods? Why not compare with these methods in this paper?