Title: Empirical software metrics for benchmarking of verification tools

Authors: Yulia Demyanova, Thomas Pani, Helmut Veith, Florian Zuleger

Summary:

- ***<u>Why read this paper?</u>*** This paper is important because the authors try to introduce a new way of performing software verification by building a portfolio solver. They try to show that if metrics using data-flow analysis can be extracted from the source code, then it is possible to build a machine learning algorithm which can demonstrate strong guarantees on the performance of verification tools on specific types of software.
- ***<u>Requirements:</u>*** The dataset used for the results in this paper are benchmarks from SV-COMP 2012.
  The metrics used in this paper fall under these 3 categories:
  (1) Program variables, (2) Program loops, (3) Control flow

  The machine learning algorithm used is support vector machines (SVMs) with RBF kernel. The authors conclude that their technique outperforms all other tools.

- ***<u>How does SV-COMP work:</u>*** A verification task in SV-COMP is a c source file and a verification property. The expected answer is provided for each task and these tasks are partitioned into categories, manually grouped by characteristic features. The competition assigns score to each tool's result on a given task. A category score and overall score is computed based on how the tool performs on benchmarks from each category and overall on all the given tasks.
- ***<u>Formulating the problem:</u>*** The above described process is formulated into a machine learning as follows:
  Given a c file, a feature vector (task) is a triple <f,p,type> such that f is a vector representation of metrics defined above, p is the property and type is the property type.
  ExpAns: Tasks -> {true, false} is a mapping where given a property and task, it is true if the property holds for a given task and false otherwise.
  Each tool with the above mapping returns a result as <answer, runtime> where answer can be <true, false, unknown>. The final output is the best tool which satisfies the 2 conditions:
  1. Returns the answer = ExpAns (most correct answers)
  2. Has the minimum (overall) runtime.
- ***<u>Details about the problem:</u>*** For each verification tool, for a given task and a property, it can have 3 cases: (a) ExpAns = returned answer, (b) returned answer = Unknown, (c) returned answer = incorrect answer. The tools are scored based on their outcomes.
- ***<u>More work:</u>*** The authors refined the weight matrix after observing in SV-COMP 2014 that *false* bugs occur in less than 4% of all runs (on all tools) and changed the scoring (penalizing more for false bugs) after these observations.
- ***<u>Results:</u>*** After making the modifications to the scoring system and weight matrix of the algorithm, they observed that the technique indeed outperforms other tools both in terms of score and runtime. The results were compared with 8 other tools including the top 3 from SV-COMP 2014and SV-COMP 2015.

Discussion:

- **Advantages:**
  1. The paper succeeds in highlighting the importance of using a portfolio solver in software verification and successfully highlights that these techniques can be a possible future of software verification.
  2. The algorithm does achieve the best scores and runtime thus highlighting that these techniques can not only give correct results but can do that faster.
  3. I think that the idea was neat and considering that it was an early work, it definitely made me curious on how these techniques can be further refined.
  4. The approach does give insights on how there can exist some similarity/ pattern between control flow of different c programs.
- **Disadvantages:**
  1. The success of such algorithms strongly depends on the metrics that are extracted from the source code and there is little known about what metrics can be considered useful.
  2. Since machine learning techniques are highly unpredictable in terms of controlling the error rate, the authors had to do an exhaustive search to find optimal parameters with respect to error measure function.
  3. Even though there is a speedup in runtime using this technique, it is important to note that there was a lot of pre-processing on the data like generating the feature vectors from the c files. The authors do not mention whether the time required to extract these feature vectors is included in the runtime or not.
  4. The authors tried this experiment with a fixed ML algorithm and do not compare the performance with other machine learning algorithms and provide no specific reasoning on why they decided to choose non-linear SVM.
  5. The authors do not mention how many false bugs occurred in their tool and the modified scoring system doesn’t necessarily facilitate that higher the overall score, less are the false bugs.
  6. Although the algorithm seems novel and practical, I personally think that the benchmark suite available at this point is not enough to build an accurate model. Although I think that a tool like this can be used to avoid manually classifying the tasks into categories.
  7. The authors mention using benchmark suite from SV-COMP 2012 but in their results, show performance improvement on tools presented in SV-COMP 2014 and SV-COMP 2015. Does this mean they updated their benchmark suite? Or did they used the old benchmark suite against these tools?

Overall, I think it was a good read and I strongly recommend to read this paper to understand the math and implementation details.