Quantitative Formal Methods

Modelling Speedoo

Renan Greca

December 11, 2018

Introduction

Speedoo, introduced by Chen et al. [1], is a tool designed to help software developers find optimization opportunities in large-scale programs. Instead of just outputting a list of methods that should be optimized, the core of the Speedooo solution lies on the concept of *optimization spaces*, each of which is a set of related methods that together provide one optimization opportunity.

For example, if two methods f and g are reasonably fast in isolation, a traditional tool might not suggest them as optimization opportunities. However, if f calls g multiple times during execution, f and g form an optimization space and should be analyzed by the developers together – perhaps there is a way of optimizing the number of calls to g within f.

This document describes the process of modelling Speedoo using a petri net, which provides some insight regarding the running times of the steps taken by an execution of Speedoo. Furthermore, it also provides a comparison with another optimization-suggesting tool, YourKit, using discrete-time markov chains.

Modelling Speedoo's petri net

To analyze Speedoo, a petri net was designed using the PIPE tool [2]. The steps taken by the tool were used as a general guide of the places and transitions of the network. Figure 2 from the Speedoo article provides the steps, complemented by Table 8, also from Speedoo, which provides the performance overhead of each step.

The most relevant step for this analysis is the first one, entitled Metric Computation. For Speedoo, the authors use three separate tools:

1. Titan, which computes *Architectural Metrics* (AM);
2. Understand, which computes *Static Complexity Metrics* (SCM); and
3. YourKit, which computes *Dynamic Execution Metrics* (DEM).

Here, we assume the there tools run in parallel, so their computations are modelled as such in the petri net.

Furthermore, the Speedoo article provides the time each tool takes to compute metrics, and these times were used to assign rates to the petri net transitions. Speedoo was tested on three separate projects, so the average time was taken for each computation (we can interpret this as "average running time for a project of arbitrary size"). Then, these times were normalized and subtracted from 1, so longer times approach 1 and shorter times approach 0. A simple algorithm was developed to quickly retrieve these values.

The other two major steps of Speedoo, Method Priotization and Optimization Space Localization, were each modelled as a single step, for two reasons: first, their sub-steps are dependent of each other, so they must be executed linearly and cannot take advantage of parallelism; and second, the paper only provides the total running time of these steps.

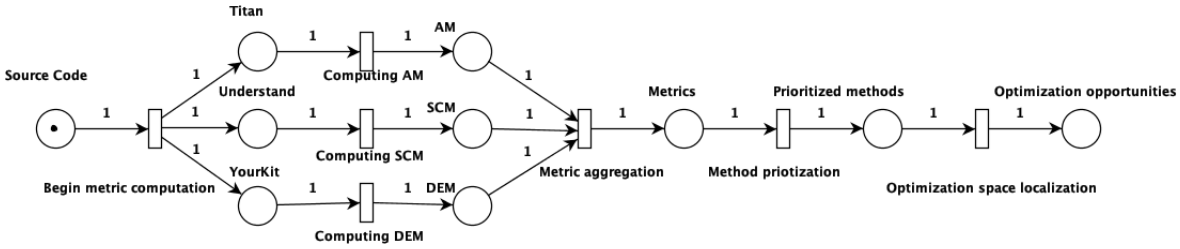

Figure 1: The petri net describing Speedoo

Figure Figure 1 shows the petri net that describes the steps taken by Speedoo. An execution of the petri net is as follows:

1. Speedoo starts at an initial place which contains the input project's source code;
2. Then, this source code is duplicated and sent to the three tools for metric computation;

	Src. code	Titan	Under.	YourKit	AM	SCM	DEM	Metrics	Pr. meth.	Opt. opp.
M0	1	0	0	0	0	0	0	0	0	0
M1	0	1	1	1	0	0	0	0	0	0
M2	0	1	1	0	0	0	1	0	0	0
M3	0	1	0	1	0	1	0	0	0	0
M4	0	0	1	1	1	0	0	0	0	0
M5	0	1	0	0	0	1	1	0	0	0
M6	0	0	1	0	1	0	1	0	0	0
M7	0	0	0	1	1	1	0	0	0	0
M8	0	0	0	0	1	1	1	0	0	0
M9	0	0	0	0	0	0	0	1	0	0
M10	0	0	0	0	0	0	0	0	1	0
M11	0	0	0	0	0	0	0	0	0	1

Table 1: The reachable states of the petri net

3. Each tool computes the appropriate metric;

4. Once all metrics are computed, they are sent back to Speedoo;

5. With the computed metrics, the Method Priotization step is performed;

6. With the prioritized methods, the Optimization Space Localization step is performed, producing the desired output.

With this, we can run analysis on the petri net and find out some information about it. Table 1 shows the reachable states of the petri nets: each column is a place and each row is a state in which places have a certain number of tokens. We can see that, due to the parallel computation of metrics, states M1 through M8 represent all the possible permutations of metrics being computed, depending on which tools performs its analysis faster.

However, despite all these states being reachable, they are not equally likely. This happens because the Dynamic Execution Metrics take much longer to compute than the other metrics – making it so step M7 is the one in which the model spends the most time. Indeed, since the computation of Dynamic Execution Metrics takes hours instead of seconds, the time spent waiting for YourKit is several orders of magnitude greater than anything else in the model.

Due to the relative simplicity of the petri net model, not much else can be extracted from the analysis. The program always terminates and produces an output, and there are no cyclic paths, so the time spent on M11, the final state, tends to infinity. In this case, waiting longer for the

State	Value
M0	1
M1	0.50018
M2	0.5002
M3	0.9999
M4	1.0007
M5	1
M6	1.0008
M7	10,000
M8	1
M9	1.0002
M10	1.00432
M11	∞

Table 2: Time spent in each state of the petri net

Dynamic Execution Metrics is rather obvious from the data – it's thousands of times slower than other operations. However, if the times were more comparable, this analysis could be an interesting way of detecting bottlenecks in the process.

Modelling Speedoo's markov chain

Another analysis of Speedoo was performed using discrete-time markov chains, which allow us to determine how many iterations it would take for a value to reach a steady state. For the purposes of this work, we used information provided in Table 6 of the Speedoo article, which details the density of suggested methods that were actually optimized in the projects the authors considered. To model and run the markov chains, the Octave tool was used [3].

Here, we consider two such projects: Avro and PDFBox. Assuming that, for every new release of a project, the same percentage of suggested methods are optimized, we want to find out how many releases it would take for all the suggested methods to be optimized. Furthermore, we also assume that, once a method has been optimized, it is never suggested again.

In all figures, the red points represent the proportion of suggested methods that were optimized, while the blue points show the proportion of methods that still need to be optimized.

As such, Figure 2 shows that, using Speedoo, it would take around 50 releases for Avro to optimize all the methods suggested. PDFBox, on the other hand, would take only 12, shown in Figure 3.

The alternative to Speedoo investigated by the authors, YourKit, proves much worse using the same criteria: Avro would take over 140 releases to be optimized (Figure 4) while PDFBox would take nearly 40 releases (Figure 5).

However, in order to achieve these results, many assumptions about the projects had to be made. Realistically, software optimization does not evolve at a constant pace and it cannot be expected that all methods are truly optimizable.

Therefore, although this analysis provides an interesting comparison between Speedoo and YourKit, it does not provide a concrete measure of Speedoo's efficacy.

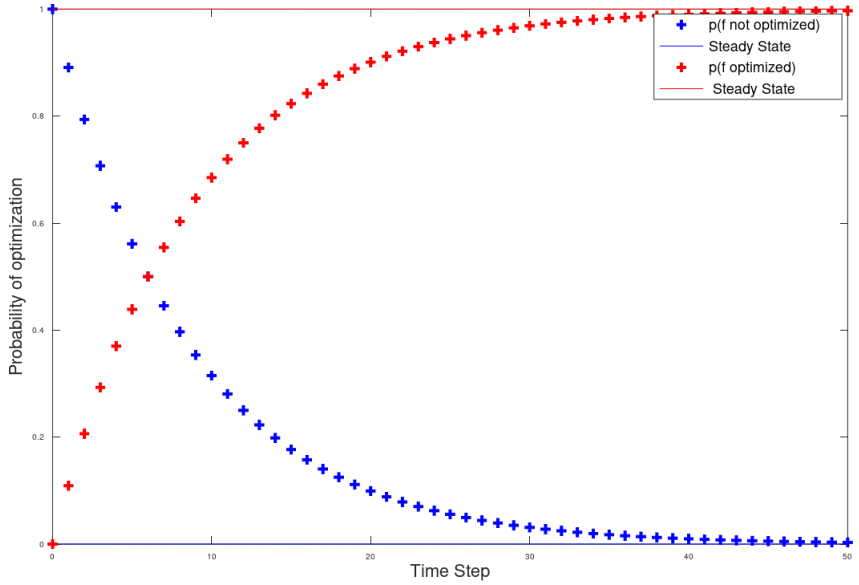

Figure 2: Running Speedoo on the Avro project

Conclusion

In this work, the Speedoo optimization tool was analysed by modelling its steps and results using a petri net and markov chains.

The petri net and the markov chains used here are just an interpretation of Speedoo, using information available on the article that presents the tool. Other interpretations, possibly with different sets of information, could generate different representations and even different analyses of Speedoo.

The models introduced here succeed in providing us with two pieces of information: first, the

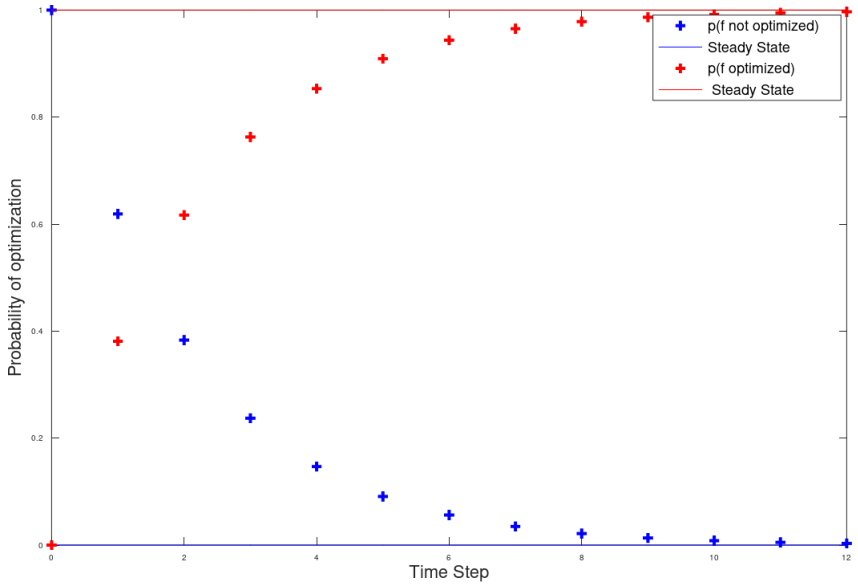

Figure 3: Running Speedoo on the PDFBox project

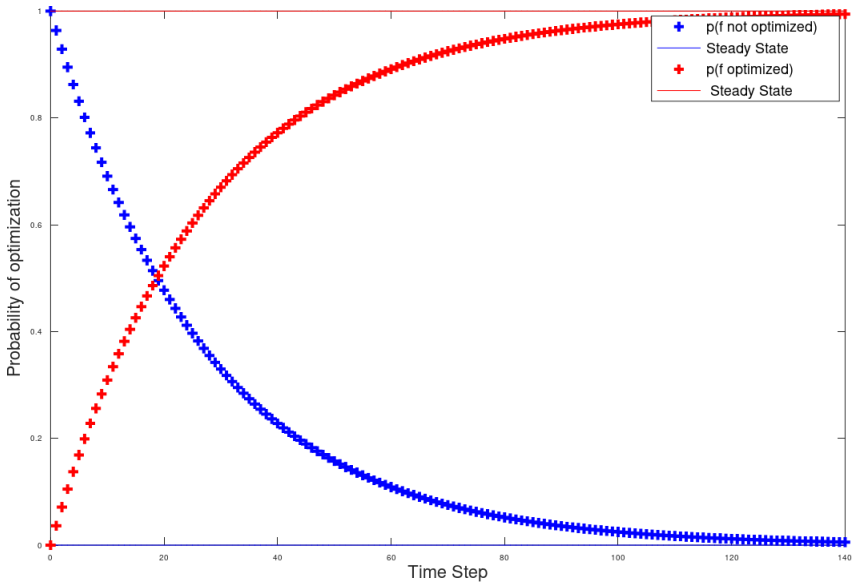

Figure 4: Running YourKit on the Avro project

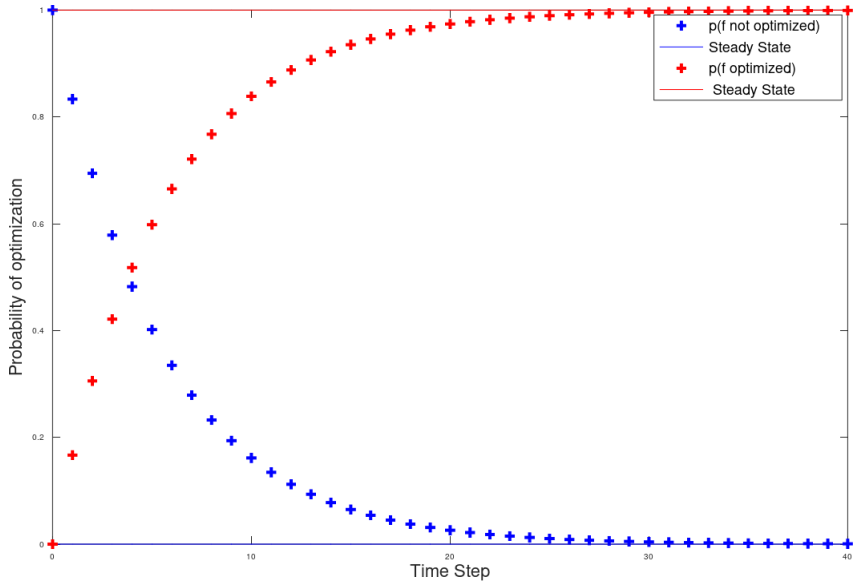

Figure 5: Running YourKit on the PDFBox project

petri net clearly shows the bottleneck created by the Dynamic Execution Metrics computation; and second, the markov chains show how much longer it would take to optimize a software project using Speedoo's alternative YourKit.

References

[1] Chen Z, Chen B, Xiao L, Wang X, Chen L, Liu Y, Xu B (2018), "Speedoo: Prioritizing Performance Optimization Opportunities," *Proceedings of the 40th International Conference on Software Engineering - ICSE '18.*

[2] Department of Computing, Imperial College London (2002), "The Platform Independent Petri net Editor PIPE," `http://pipe2.sourceforge.net`.

[3] Eaton JW (1993), "GNU Octave," `https://www.gnu.org/software/octave/`.