Responses to Reviewer Comments

Reviewer #1:

I like the basic ideas of taking theoretical ideas to the test and perform actual experiments. However, I fail to see the significance in this paper. The theoretical results are nicely stated, but follow obviously from the tree-structure of the refinement relation. The algorithms used in the evaluation are simple and not really motivated. Why should we expect different behavior based on the different path selection methods? Given a small number of dead features, the result (use leaf-first checks) is obvious, as this will normally show the leaf and hence the whole path to be non-dead without the need for further checks.

RESPONSE: Intuitively we should expect different path selection policies have different efficiency. An example is shown below, in Figure 1. Suppose the original feature model contains no dead features, and the example shows the non-determined features left after the first iteration of path selection and checking. We can see that they have totally different effects. Given that a real feature model only contains a few dead features, in other words, most part of it has no dead features, so we should expect different behaviors of the policies in such cases.

To the second question, we have two leaf-first algorithms (binary and linear), so we still need to compare them through experiments.

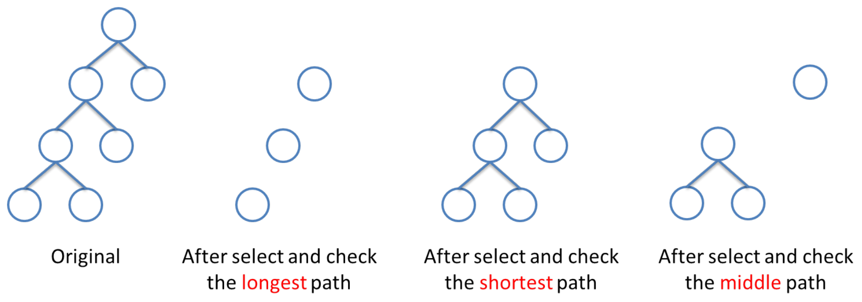
Original
After select and check the longest path
After select and check the shortest path
After select and check the middle path

Figure 1. The non-determined features left after the first iteration of our algorithms.

What I'm missing completely is an analysis of the relevance of the approach. SAT is NP-complete and notoriously finicky. Hence I would expect the SAT solver to show very different run times for the different problems. In this paper, the authors seem to consider the run time of the SAT solver to be constant - is that realistic? There also is no analysis of the cost of implementing the different path selection strategies (given that the influence is minor, would "pick any open leaf and use the longest path of undetermined features from there" not be an obvious approach)?

RESPONSE: The proposed algorithms can *determine* the deadness of *all* features by *checking* only *some* of the features. Checking a feature means invoking a SAT solver for the feature. In our paper, we compute the number of such invocations of SAT solvers (i.e. the number of checked features), not the running time of the solvers. However, in the current version of our paper, we designed more detailed experiments including a comparison of *running time* of the algorithms. Please see the revised Section 4 for details.
To the second question, different path selection policies take exactly the same time to select a path (see Line 5 to Line 7 in Algorithm 3, pg. 11), so there is no need to compare their costs.

I am missing an explanation of how the real feature models are imbued with 0-4 non-trivial dead features.

RESPONSE: Explanations and an example are added to Section 4.1 and Figure 4.

Page 3, Definition 1: Your definition of Refine via its inverse relation is elegant, but a bit confusing. Also, it's not quite correct, as you do not specify that the refinement graph is connected - as far as I can see, a forest would be compatible with your definition.

RESPONSE: The definition has been revised to ensure that the features form a tree.

The detailed listing of the sub-algorithms staring on page 10 uses up a lot of space not really necessary. Most of them are obvious, and adequately described with one or two sentences of text.

RESPONSE: The descriptions of the algorithms have been reduced to less than one page while not affecting the understandability.

Reviewer #2:

Overall I found that the paper takes to much space for listing the algorithms and does not make enough out of what is important, the results. What does the analysis actually tell me?

RESPONSE: The descriptions of the algorithms have been reduced to less than one page. The experiments are extended as well as the analysis (see Section 4).

*If I understand correctly, then your discussion of efficiency is based on the "times the SAT solver is called", right? What you do not take into account is the *time* it* *takes to execute your own algorithms, e.g., the calculation of find_non_determined_roots and find_leaves_under in Algorithm 3. How does the picture change if you take that into account?*

RESPONSE: We have extended the experiments to include comparisons of execution time of the algorithms (you may want to see Figure 5, Section 4).

It might be interesting from an academic point of view, but you did not convince me that the problem is actually relevant in practice. How long does it take to determine dead features in a realistic feature model? Is that a problem? Do people care?

RESPONSE: Real feature models can be very large. For example, the Linux Core feature model [She et al. 2010] has nearly 6000 features and more than **9000** constraints. There are also many real feature models that have non-Boolean constraints (i.e. more complicated than propositional logic). Therefore generally, it takes a lot of time for professional solvers to check these feature models. In addition, to check the deadness of a **single** feature you have to check the satisfiabiity of the **whole** logic formula representing the feature model (i.e. check if the formula "*whole_feature_model* AND *the_feature*" is satisfiable), so the situation is even worse for detecting dead features. That is why we focus on reducing the number of checked features (thus reducing the number of such satisfiability checks) in this paper.

(*Reference*: She, S., R. Lotufo, T. Berger, A. Wasowski, and K. Czarnecki, "Variability Model of the Linux Kernel", Fourth International Workshop on Variability Modeling of Software-intensive Systems (VaMoS 2010), Linz, Austria, 2010.)

Is "leaf first" the same as "depth-first search"? Please clarify.

RESPONSE: Not exactly. For a single refinement path, a classic depth-first search would check the *parent* first, and then its child, while the leaf-first search checks the *child* first, and then its parent.

*You need to explain Table 4 better, because that is you main result. What do the numbers mean? What units have the numbers? These are times, i.e. how *often* you had to invoke the SAT solver? Why the fractions? How often did you run the experiment?*

RESPONSE: They show how **often** a SAT solver is invoked, in **average**. That's why we have fractions. See the extended explanation in Section 4.1 as well as an example shown in Figure 4.

Did you think about averaging the values in columns/rows in Table 4?

RESPONSE: See previous response. The values are already average values.

In the related work section "(not a or b) and (not a or b)", some misunderstanding here?

RESPONSE: We are sorry for the mistake, it is actually (not a or b) and (not a or not b).