# EXIST: Exploitation/Exploration Inference for Statistical Software Testing [*]

**Nicolas Baskiotis, Michèle Sebag, Marie-Claude Gaudel, Sandrine Gouraud**
LRI, Université de Paris-Sud, CNRS UMR 8623
Bât.490, 91405 Orsay Cedex (France)
{nbaskiot, sebag, mcg, gouraud}@lri.fr

## Abstract

Path-based Statistical Software Testing is interested in sampling the feasible paths in the control flow graph of the program being tested. As the ratio of feasible paths becomes negligible for large programs, an ML approach is presented to iteratively estimate and exploit the distribution of feasible paths.

## 1 Introduction

This paper is motivated by Statistical Software Testing (SST) and more precisely path-based SST [9]. Path-based SST considers the control flow graph of the program being tested, and samples the feasible paths in this graph (i.e. such that there exists an input case exerting the path); a test set is constructed by gathering all input cases associated to the feasible path samples. However, as the control flow graph is an overly general description of the program, the fraction of feasible paths might be tiny, and it is most often negligible for large program sizes (up to $10^{-5}$ or $10^{-15}$).

This paper presents a generative learning approach aimed at sampling the feasible paths in the control flow graph. This approach, called *EXIST* for *Exploration - eXploitation Inference for Software Testing*, is inspired by both Estimation of Distribution Algorithms (EDAs) [2] and Online Learning [1, 6]. *EXIST* proceeds by iteratively generating candidate paths based on the current distribution on the program paths, and updating this distribution after the path has been labelled as feasible or infeasible. *EXIST* was made possible by the use of an original representation, extending the Parikh map [12] and providing a powerful propositional description of long structured sequences (program paths). Another original contribution, compared to on-line learning [6, 13] or reinforcement learning, is that our goal is to *maximise the number of* distinct *feasible paths* found, as opposed to learning a concept or a fixed policy.

The paper is organised as follows. Section 2 introduces the formal background and prior knowledge related to the SST problem, and describes the extended Parikh representation. Section 3 gives an overview of the *EXIST* system. Section 4 describes the experimental setting and goals, and reports on the empirical validation of the approach on real-world and artificial problems. The paper concludes with some perspectives for further research. Related work are briefly reviewed in Appendix A.

## 2 Prior knowledge and Representation

The control flow graph of the program being tested is a Finite State Automaton (FSA) based on some finite alphabet $\Sigma$, where $\Sigma$ includes the program nodes (conditions, blocks of instructions), and the FSA specifies the transitions between the nodes (Fig. 1). A program path is a finite length string on $\Sigma$, obtained by iteratively choosing a node among the successors of the current node until the final node noted $v_f$ is found.

[*]The first two authors gratefully acknowledge the support of `Pascal` Network of Excellence IST-2002-506 778.