A Related Work

Although Program Synthesis is among the grand goals of Machine Learning, to our best knowledge the application of Machine Learning to Software Testing (ST) has seldom been considered in the literature.

Ernst et al. [10] aim at detecting program invariants, through instrumenting the program at hand and searching for predetermined regularities (e.g. value ranges) in the traces.

Brehelin et al. [5] consider a deterministic test procedure, generating sequences of inputs for a PLA device. An HMM is trained from these sequences and further used to generate new sequences, increasing the test coverage.

In [15], the goal is to test a concurrent asynchronous program against user-supplied constraints (model checking). Grammatical Inference is used to characterise the paths relevant to the constraint checking.

Xiao et al. [16] aim at testing a game player, e.g. discovering the regions where the game is too easy/too difficult; they use active learning and rule learning to construct a model of the program. A more remotely related work presented by [18], is actually concerned with software debugging and the identification of trace predicates related to the program misbehaviours.

In [10, 15], ML is used to provide better input to ST approaches; in [5], ML is used as a post-processor of ST. In [16], ML directly provides a model of the black box program at hand; the test is done by manually inspecting this model.

B Empirical validation and sensitivity analysis

Figs. 2, 3 and 4 respectively display the average number of feasible paths out of 10,000 path generations on Easy, Medium and Hard problems.

Fig. 5 displays the average performance of *EXIST* for various numbers of feasible and infeasible paths.

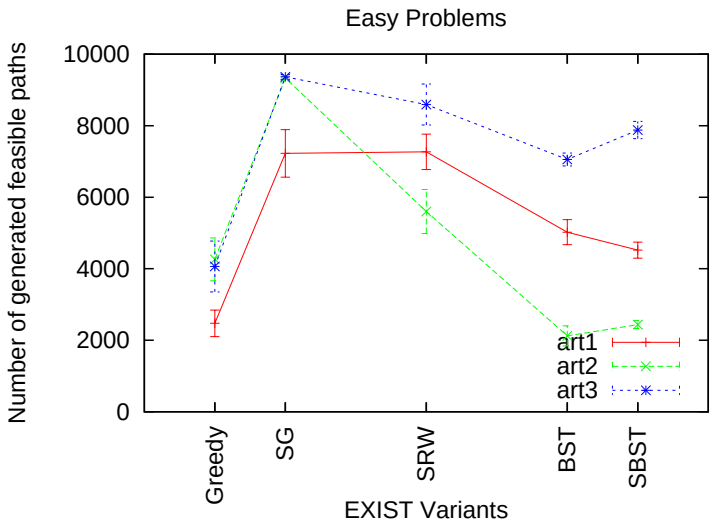

Figure 2: Number of distinct feasible paths generated by *EXIST* out of 10,000 trials on Easy problems, starting from 50 feasible/50 infeasible paths, averaged on 10 independent runs.