Problems		Greedy	SG	BandiST	SBST	SRW
Fct4		2419 ± 84	3754 ± 612	745 ± 176	1409 ± 812	3332 ± 580
Easy	art1	2473 ± 372	7226 ± 665	5023 ± 349	4520 ± 225	7270 ± 496
	art2	4261 ± 599	9331 ± 38	2122 ±281	2439 ± 110	5600 ± 615
	art3	4063 ± 711	9365 ± 65	7056 ± 181	7879 ± 240	8592 ± 573
Medium	art4	1235 ± 333	9025 ± 118	1909 ± 358	1744 ± 853	6078 ± 479
	art5	2635 ± 497	8368 ± 149	4294 ± 1121	5106 ± 236	6519 ± 965
	art6	830 ± 187	7840 ± 448	2775 ± 1630	4588 ± 610	4920 ± 984
Hard	art7	4236 ± 292	7582 ± 217	2840 ± 95	52 ± 8	5590 ± 163
	art8	3166 ± 140	5496 ± 149	2174 ± 62	777 ± 223	1757 ± 110

Table 1: *EXIST* variants *Greedy*, *SeededGreedy* (SG), *BandiST*, *SeededBandiST*(SBST) and *SeededRouletteWheel* (SRW): Number of distinct feasible paths out of 10,000 generated paths averaged over 10 independent runs, and standard deviation.

procedure returns to state A. Otherwise (state B), the test against the infeasible paths might reject the lgg (there exists an infeasible path covered by the resulting lgg), s_t is rejected and the procedure returns to state A. Otherwise (state C), there exists no infeasible path enforcing s_t rejection and preventing the overgeneralisation of the seed set. As the seed set will from now on contain examples from different subconcepts (state C is absorbing), the probabilities estimated from the seed set are misleading and will likely lead to generate infeasible paths.

The number and quality of infeasible paths governs the transition from state B to either A (probability q) or C (probability $1 - q$). Although q should exponentially increase wrt the number of infeasible paths, it turns out that initial infeasible paths are useless to detect incorrect lggs; actually, only infeasible paths sufficiently close (wrt the Parikh representation) to the frontier of the subconcepts are useful. This remark explains why increasing the number of initial infeasible paths does not significantly help the generation process.

On the other hand, the probability of ending up in the absorbing state C (failure) exponentially increases with the number of steps of the *Seeded* procedure, i.e. the number of feasible paths. Only when sufficiently many and sufficiently relevant infeasible paths are available, is the wealth of feasible paths useful for the generation process.

Complementary experiments done with 1000 feasible vs 1000 infeasible paths show that i) the limitation related to the number of initial feasible examples can be overcome by limiting the number of feasible paths considered by the *Seeded* procedure (e.g. considering only the first 200 feasible paths); ii) in order to be effective, an adaptive control of the *Seeded* procedure is needed, depending on the disjunctivity of the target concept and the presence of "near-miss" infeasible paths in $\mathcal{E}$.

5 Conclusion and Perspectives

The presented application of Machine Learning to Software Testing relies on an original representation of distributions on strings, coping with long-range dependencies and data sparsity. Further research aims at a formal characterisation of the potentialities and limitations of this extended Parikh representation (see also [7]), in software testing and in other structured domains. The second contribution of the presented work is the *Seeded* heuristics inspired by [14], used to extract relevant distributions from examples representing a variety of conjunctive subconcepts. This heuristics is combined with Exploration vs Exploitation strategies to construct a flexible sampling mechanism, able to retrieve distinct feasible paths with high probability. With respect to Statistical Software Testing, the presented approach dramatically increases the ratio of (distinct) feasible paths generated, compared to the former uniform sampling approach [9].

Further research aims at estimating the distribution of the feasible paths generated by *EXIST*, and providing PAC estimates of the number of trials needed to reach the feasible paths (hitting time). In the longer run, the extension of this approach to related applications such as equivalence testers or reachability testers for huge automata [17] will be studied.