Lattice-Search Runtime Distributions May Be Heavy-Tailed

Filip Železný[1], Ashwin Srinivasan[2], David Page[3]

[1] Dept. of Cybernetics
Faculty of Electrical Engineering
Czech Technical University
Karlovo Nám. 13, 121 35 Prague, Czech Republic
zelezny@fel.cvut.cz
[2] Oxford University Computing Laboratory
Wolfson Building, Parks Road
Oxford OX1 3QD, UK
ashwin@comlab.ox.ac.uk
[3] Dept. of Biostatistics and Medical Informatics and Dept. of Computer Science
University of Wisconsin
1300 University Ave., Rm 5795 Medical Sciences
Madison, WI 53706, USA
page@biostat.wisc.edu

Abstract. Recent empirical studies show that runtime distributions of backtrack procedures for solving hard combinatorial problems often have intriguing properties. Unlike standard distributions (such as the normal), such distributions decay slower than exponentially and have "heavy tails". Procedures characterized by heavy-tailed runtime distributions exhibit large variability in efficiency, but a very straightforward method called *rapid randomized restarts* has been designed to essentially improve their average performance. We show on two experimental domains that heavy-tailed phenomena can be observed in ILP, namely in the search for a clause in the subsumption lattice. We also reformulate the technique of randomized rapid restarts to make it applicable in ILP and show that it can reduce the average search-time.

1 Introduction

In the recent paper [4], Gomes et al. observe that procedures for solving propositional satisfiability problems exhibit a remarkable runtime variability. The runtimes vary greatly depending on the choice of a particular heuristic, a given problem instance, or - for stochastic methods - on the choice of different random seeds (initial truth assignments), or on another source of randomness. Often a satisfiability procedure "hangs" on a given problem instance, while a different stochastic run solves the same instance quickly. Even for a deterministic procedure and a given problem

instance, small amount of randomization (e.g. in the employed heuristic) yields again largely varying search-costs, some of which are substantially lower than that of the deterministic algorithm.

In their empirical study it is shown that distributions of the runtimes of many search algorithms decay slower than exponentially and asymptotically have *heavy-tails*. Unlike standard probability distributions, such as the normal distribution, where events that are several standard deviations from the mean are very rare, in heavy-tailed distributions there is a non-negligible probability that an event with an extremely high cost occurs. For example, in one of the studied problems, 80% of the runs solve the problem in 1,000 backtracks or less, however 5% of the runs do not result in a solution even after 1,000,000 backtracks. Gomes et al. believe that the heavy-tailedness is a property of many exhaustive backtrack algorithms for solving hard combinatorial problems, and offer a technique called *randomized rapid restarts* that exploits this property in order to reduce the average search-time. The technique was used to find solutions of previously unsolved instances of hard combinatorial problems.

Although many search problems give rise to a heavy-tailed distribution, others do not [5]. Our aim is to find out whether heavy-tailed runtime distributions occur in ILP. Namely, we empirically study the runtimes of the search for a first-order clause with defined desired properties, in the lattice imposed by the subsumption relation. Furthermore we reformulate the randomized rapid restarts algorithm to be applicable on the ILP search problem and on two important ILP benchmarks we evaluate whether it reduces the search-cost with respect to a deterministic exhaustive search.

The following section defines formally the notion of a heavy-tailed distribution and describes the method of randomized rapid restarts. Since the method requires randomization of the exhaustive search, Section 3 describes our way of randomization of the lattice search, based on the selection of a random starting clause (seed). The core of the study is Section 4 which will test empirically the hypothesis that heavy-tailed runtime distributions describe the clause lattice-search using benchmark ILP problems. In the same section, we shall also apply the technique of randomized rapid restarts on the same domains, and investigate whether it improves search efficiency. We summarize our observations in Section 6.

2 Heavy-tailed Distributions and Randomized Rapid Restarts

The cumulative probability distribution $Pr(X < x)$ of a random variable X is a non-decreasing real function on the real interval $-\infty < x < \infty$ and will be denoted $F(x)$, i.e. $Pr(X > x) = 1 - F(x)$. Standard probability distributions have exponentially decreasing tails, e.g. for the standard normal distribution F_n it holds that

$$(1 - F_n(x)) \sim \frac{1}{x\sqrt{2\pi}} \exp \frac{-x^2}{2} \tag{1}$$

where $g(x) \sim h(x)$ denotes $\lim_{x\to\infty} g(x)/h(x) = 1$.

Recently, in the area of algorithms for hard combinatorial problems [4] but also other areas such as statistical physics, economics etc., different phenomena have been shown to obey heavy-tailed distributions which often lead to non-intuitive behaviour. For this class of distribution it holds that

$$(1 - F(x)) \sim Cx^{-\alpha}, x > 0 \tag{2}$$

where $0 < \alpha < 2$ and $C > 0$ are constants. It is remarkable [4] that such distributions have finite mean but no finite variance if $1 < \alpha < 2$. If $\alpha \leq 1$, the distribution has even neither a finite mean nor a finite variance.

To determine whether a distribution estimated by a series of measurement has a heavy-tailed nature, i.e. it does not decay exponentially, we plot the measured distribution values in a diagram with both axis logarithmically scaled, because an exponentially decreasing distribution should show a faster-than-linear decay in the log-log scale. For example, substituting x with $\exp(x)$ in the normal distribution decay (see Eq. 1) and taking log yields

$$\log\{1 - F_n(\exp[x])\} \sim -\left(x + \frac{\exp(2x)}{2}\right) \tag{3}$$

while the same operation on the heavy-tailed distribution decay (see Eq. 2) yields $-\alpha x$, i.e. a heavy-tailed distribution should exhibit an approximately linear decay on the log-log scale as x approaches infinity.

Let us now consider an exhaustive-search algorithm randomized in such a way, that it starts the search at a randomly chosen point of the search space (seed). Depending on the particular type of a search problem and algorithm, the distribution of times required to reach a solution from

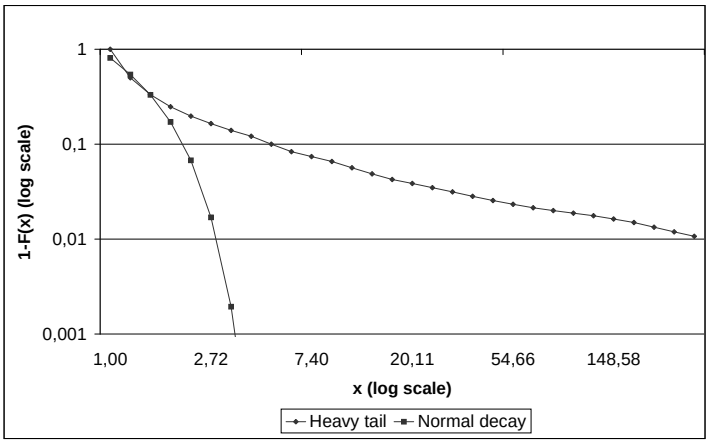

Fig. 1. Example of normal and heavy-tailed distributions on a log-log scale. The normal distribution decays faster than linearly while the heavy-tailed distribution decay shows an approximately linear decay.

such seeds may or may not be heavy-tailed.[1] If a heavy tail is observed, the runtime variance and mean may be infinite and there is a non-negligible probability that a chosen seed will start an extremely costly search, although many other seeds may produce a quick path to the solution. A direct way to reduce the variance and mean in such a case is to run the exhaustive search up to a certain cutoff point and then restart at a different seed if a solution is not found. Clearly, this approach called *randomized rapid restarts* avoids the algorithm from getting trapped in a very costly path and exploits the high chance of obtaining an essentially shorter path in the next trial. It is shown in [4] that the randomized rapid restarts technique is superior to deterministic exhaustive searches in many propositional domains.

[1] Gomes et al [5] discover the heavy-tailedness of different search problems purely empirically and report that further studies are needed to determine exactly what characteristics of combinatorial search problem lead to heavy-tailed behaviour. In this ILP-focused study we also take an empirical approach.

3 Randomizing the Lattice Search

We consider the normal ILP problem [10], namely we assume the sets of positive (negative) examples E^+ (E^-) and background knowledge B. We also assume that a search lattice of legal clauses has been defined by the generality (subsumption) relation, a clause mode language $\mathcal{L}$, and bounded by the most specific element $\perp$ (the bottom clause). This is a usual assumption of ILP systems based on the concept of inverse entailment [9], such as Aleph [6] and Progol [9].

The standard approach of conducting the lattice search is to start with the most general (most specific, $\perp$) clause and then proceed in a top-down (bottom-up) manner. However, starting the search with a different clause from the interior of the lattice may lead to a shorter runtime needed to reach the desired clause. Our plan is to investigate the distribution of such runtimes when the starting clauses are selected randomly from the lattice. If this distribution proves to be heavy-tailed, we will be able to utilize the technique of randomized rapid restarts to avoid the extreme-cost paths and improve the average performance.

The randomized search algorithm proceeds as follows. Some number of times (maxtries), the algorithm will carry out a short search (bounded by maxtime, Section 4). Each search begins by stochastic selection of a starting clause. The search is a deterministic best-first search, with heuristic function $h = pos(C) - neg(C)$. Here $pos(C)$ is the number of positive examples deducible from $C \wedge B$[2], and $neg(C)$ is analogous for negative examples. From a given clause C, the neighbors of C in the search space are defined by a nontraditional refinement operator ρ. It differs from usual refinement operators employed in the top-down search of the mentioned systems in that it produces the set $REFS = \rho(C)$ of all neighbours of C in the lattice, i.e. also including clauses that subsume C. Therefore this kind of search can be seen as radial, rather than top-down or bottom-up (visualized in Fig. 2). With this refinement operator, all nodes in the lattice can be reached from any starting node.

We shall now address the problem of choosing the seed, i.e. the initial stochastic clause selection. The principal difficulty of its implementation lies in devising a procedure for uniform random sampling of clauses from the search space. Here, we describe a procedure (from [14]) that does not require prior generation of all elements of the search space. Recall that these are definite clauses obtained from subsets of literals drawn from a

[2] In the case when C is constructed as an extension of an existing partial theory H in a greedy cover search, we assume that H has been added to B.

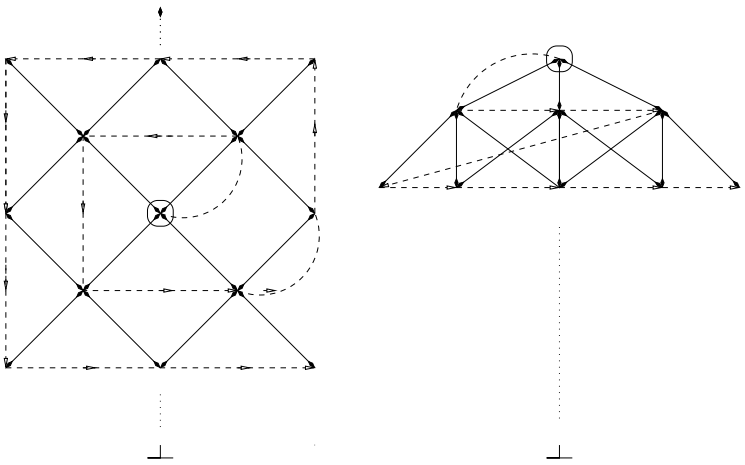

Fig. 2. A schematic visualization of the radial lattice search (left) compared to a top-down search (right). Nodes are explored starting at the encircled point and then following the dashed line. This view is simplified in that the employed heuristic function $h(C)$ is assumed to be constant for all C and descendants of explored nodes are inserted in the end of the *open* list, which in the top-down case corresponds to a breadth-first search.

most specific (definite) clause $\perp$. Additional provisos are that each subset is of cardinality at most $c + 1$ (where c is a user-specified maximum number of negative literals) and is in the language $\mathcal{L}$. Let $\mathcal{C}$ denote all such clauses. Further, let the number of clauses in $\mathcal{C}$ with exactly l literals be n_l and $\mathcal{N}$ denote the subset of natural numbers $\{1, \ldots, |\mathcal{C}|\}$. Define a function $h : \mathcal{C} \rightarrow \mathcal{N}$ such that $h(C) = \sum_{i=1}^{|C|-1} n_i + j$ where $|C|$ is the number of literals in C and $1 \leq j \leq n_{|C|}$. That is, h provides a sequential enumeration of clauses by length. While many functions fit this requirement (depending on the enumeration adopted), it is easy to show that any such h is both $1-1$ and *onto*. It follows that h is invertible – that is, given a number in $\mathcal{N}$, it is possible to find a unique clause in $\mathcal{C}$ provided the n_i (and c) are known. In principle, it is therefore possible to achieve the selection required by randomly choosing a number n in $\mathcal{N}$ and returning $C = h^{-1}(n)$. Such an inverse function works as follows. Given a number $n > 0$: (a) find the largest number $l = 0 \ldots c$ such that $j = n - \sum_{i=0}^{l} n_i > 0$; (b) generate a sequence of clauses in $\mathcal{L}$ of length $l+1$. C is the j^{th} clause in this sequence. If n is randomly generated, then the clause generation process does not have to be so, and can be made more efficient by various devices. Some examples are: (a) take C to be

estimate$(\bot, \mathcal{L}, l, s)$: Given a most specific clause $\bot$ and a clause length $l > 1$, returns an estimate of the number of definite clauses of length l in $\mathcal{L}$ such that each clause is a subset of $\bot$. The estimate is obtained from a sample of size s.

1. Sample s clauses of length l from $\bot$. Each such clause consists of the positive ("head") literal in $\bot$ and a random selection, without replacement, of $l-1$ literals from the negative ("body") literals in $\bot$.
2. Determine the proportion p_l of the s clauses that are in $\mathcal{L}$.
3. return $p_l \times (|\bot| - 1) \times \ldots \times (|\bot| - l + 1)$

Fig. 3. A procedure for estimating the number of "legal" clauses of length $l > 1$. The estimate obtained in Step 3 above is unbiased [18]. The value of the sample size s needs to be decided. An option is to be guided by statistical estimation theory. This states that if values of p_l are not too close to 0 or 1, then we can be at least $100 \times (1-\alpha)\%$ confident that the error will be less than a specified amount e when $s = z^2_{\alpha/2}/(4e^2)$ [18]. Here z represents the standard normal variable as usual.

the first clause of that length (and in $\mathcal{L}$) that has not been drawn before; (b) a once-off generation of the appropriate number of clauses in $\mathcal{L}$ at each length ("appropriate" here means that the proportion of clauses of length i in the sample is $n_i/|\mathcal{N}|$); and (c) using a dependency graph over literals in $\bot$ to ensure that the random clause construction always results in clauses within the language $\mathcal{L}$.

In practice, without prior generation of the set $\mathcal{C}$, the n_i are not known for $i > 1$ and we adopt the procedure in Fig. 3[3] for estimating them.

Having constructed a method for stochastic selection of the starting clause and its subsequent deterministic refinement, we are in a position to implement the technique of randomized rapid restarts. An implementation for the clause lattice search is described in Fig. 4. The underlying principle that makes the technique applicable to the clause search is that, unlike in usual ILP approaches, we do not search through a specified number of nodes, returning the best clause found, but rather we stop the search once a clause is found meeting a pre-specified condition of 'goodness' as follows.

$$pos(C) > 1 \tag{4}$$

$$\frac{pos(C)}{pos(C) + neg(C)} > Acc \tag{5}$$

where $pos(C)$ ($neg(C)$) is the number of positives (negative) examples

[3] This method is implemented in the ILP system Aleph [6] and can serve as well for other methods of randomized local search, such as GSAT or WSAT

$rrr(Lat, Acc, B, E^+, E^-, maxtime, maxtries)$: Given background knowledge B, positive and negative examples E^+, E^-, return a clause from the given subsumption lattice Lat, that satisfies the conditions in Eqs. 4 and 5 for the given constant Acc.

1. $tries := 1$
2. Select a random starting clause C_0 using the procedure described in Section 3.
3. $searchtime := 0$, start timing.
4. Starting at C_0, perform an exhaustive radial search described in Section 3, until $searchtime > maxtime$ or a clause C satisfying Eqs. 4, 5 is found.
5. If C was found, STOP, return C.
6. If $tries < maxtries$, $tries := tries + 1$, Go to 2. Otherwise return "failure".

Fig. 4. An implementation of randomized rapid restarts for the clause lattice search. The maximum time for one exhaustive search is limited by the constant $maxtime$. The maximum number of repeated searches is $maxtries$.

covered by C. The first condition avoids the trivial solution $C = e, e \in E^+$ and the second condition is parameterized by a constant Acc.

We set the maximum number of allowed restarts $maxtries$, which should theoretically be infinite, to a finite number ($maxtries = 10$) because we cannot guarantee exclude that a clause satisfying the prespecified condition exists in the lattice. In the case of reaching the limit $maxtries$ ("failure" in Fig. 4), the positive example used to construct the current bottom clause $\perp$ is returned as the resulting clause. The setting of $maxtime$ will be discussed in connection with the experimental setting in Section 4.

Finally, to cover all of the positive examples in the training sets of the experiments, we use the greedy covering approach as usual in ILP systems, i.e. the procedure in Fig. 4 is run repeatedly with a bottom clause constructed using one selected positive, until all positives are covered, each time adding the newly constructed clause to the background knowledge and deleting the covered positives from the training set.

4 Experiments

4.1 The Aim

We shall investigate (a) if the heavy-tailed phenomenon manifests itself when searching the subsumption lattice; and (b) if utilizing RRR will help improve search efficiency for problems exhibiting the heavy-tailed phenomenon.

4.2 Materials

Our experimental material consists of two sets of pre-classified data, namely the data on the mutagenic and carcinogenic properties of some chemicals. These data are publicly available (anonymous ftp to `ftp.comlab.ox.ac.uk` in the directories `pub/Packages/ILP/Datasets/mutagenesis/aleph` and `pub/Packages/ILP/Datasets/carcinogenesis/aleph`).
We refer the reader to [16, 17] for detailed descriptions of background knowledge available for the mutagenesis task. The background information is encoded in approximately 13,000 facts. The background knowledge for the carcinogenesis problem is conceptually of a similar nature. The encoding requires approximately 24,500 facts – see [15] for more details.

All of our subsequent experiments use an Athlon 1500MHz CPU – based computer with 512KB of RAM and the ILP program Aleph (Version 3). Aleph is available at: `http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph.pl`.

The language $\mathcal{L}$ will be limited to clauses of maximum number of 4 negative literals and maximum variable depth [9] 2.

4.3 Methods and Results

We shall observe the stochastic behaviour of the randomized search-algorithm described in the previous section, namely the distribution of search-times required to find a clause C which satisfies the conditions in Eqs. 4,5, where we set $Acc = 0.7$.

Figure 5 shows the cumulative distribution $F(x)$ of runtimes for the mutagenesis task, Figure 6 an analogous distribution for the carcinogenesis task. Both of the distributions are collected from about 35,000 searches starting in random seeds. In the mutagenesis task, for example, almost 20% of runs arrive at a solution in less then 0.1s, however almost 30% of runs do not find a solution in 20s.

Figures 7 and 8 clearly show that both of the experimentally measured distributions exhibit a heavy-tail (c.f. Section 2). According to the study [4], our findings justify the application of the method of randomized rapid restarts. To use the algorithm described in Section 3, we need to set the cutoff value $maxtime$. There is no analytic way of determining the optimal value $maxtime_{opt}$ of the cutoff value, but it is reported [4] to lie below the median point of the runtime distribution $F(x)$, i.e. $maxtime_{opt} << 1s$ for both our experimental domains. As it may be infeasible in the general case to construct the runtime distributions $F(x)$ for a given problem prior to the learning process, we shall disregard the information provided by

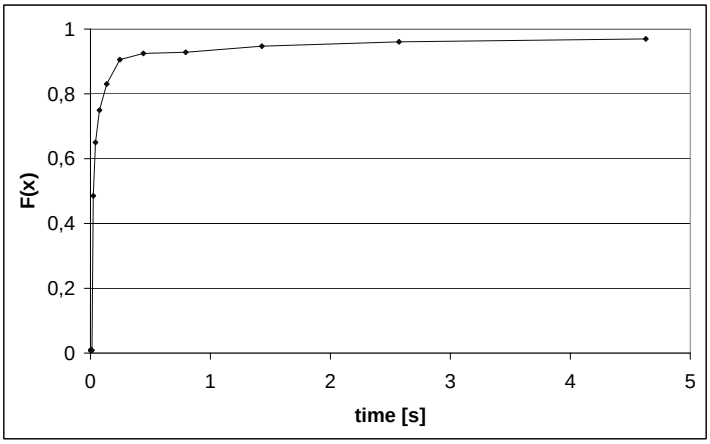

Fig. 5. The cumulative distribution $F(x)$ of runtimes of the randomized algorithm searching for a 'good' clause in the mutagenesis problem.

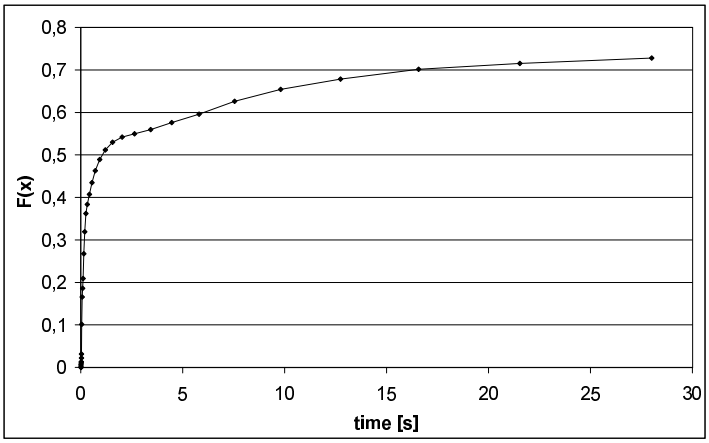

Fig. 6. The cumulative distribution $F(x)$ of runtimes of the randomized algorithm searching for a 'good' clause in the carcinogenesis problem.

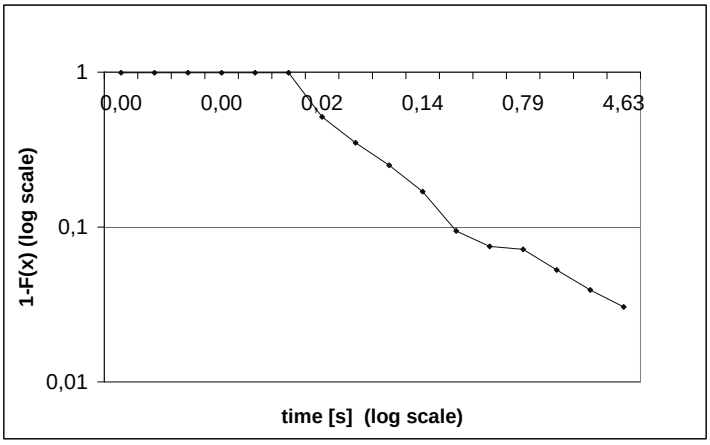

Fig. 7. A log-log plot of the distribution decay $1 - F(x)$ concerning the same data as in Figure 5.

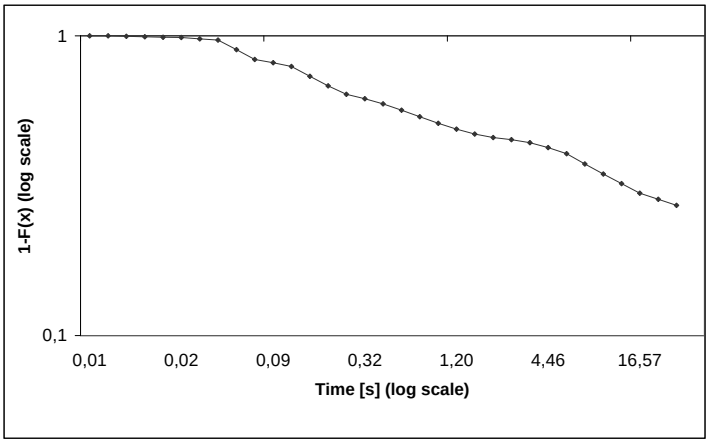

Fig. 8. A log-log plot of the distribution decay $1 - F(x)$ concerning the same data as in Figure 6.

the already generated distributions, and we choose a small *ad hoc* value $maxtime = 1s$ for both of the domains in the comparative experiments.

Table 1 summarizes the predictive accuracies and learning times of the randomized rapid restarts technique vs. the standard exhaustive breadth-first top-down search algorithm. The former method was tested with the *Acc* parameter set to the values 0.7 and 0.9. Similarly, the latter method was tested with two values 0.7 and 0.9 of the *minimum accuracy* requirement on a clause to be accepted for the constructed theory.

The results suggest that by using randomized rapid restarts we achieved a drastic reduction of the search times for the price of only a small loss in predictive accuracy.

Algorithm	MUT		CANC	
	A (%)	T (s)	A (%)	T (s)
DTD 0.7	88.76	1589	57.91	24092
	(5.99)	(461)	(9.75)	(11915)
DTD 0.9	88.23	1541	56.22	22101
	(5.63)	(459)	(8.98)	(9811)
RRR 0.7	87.71	9	54.84	74
	(7.62)	(4)	(8.97)	(10)
RRR 0.9	86.31	24	57.57	126
	(8.67)	(10)	(6.39)	(71)

Table 1. Estimated predictive accuracies (A) and theory construction times (T). The entries are from a 10-fold cross-validation design with time entries rounded up to the nearest second. The numbers in parentheses are estimates of standard deviation. These are obtained by a simple binomial formula that ignores the dependencies across cross-validation runs. Exact calculation of standard deviations for results from cross-validation designs is confounded by these dependencies but the approximation used here has been found to be adequate (see [1], pg 307). The algorithms result from two search techniques employed by Aleph, namely: deterministic top-down (DTD) (with minimum clause accuracy setting 0.7 and 0.9, respectively) and randomized rapid restarts (with clause threshold 0.7 and 0.9, respectively). MUT refers to the mutagenesis problem, CANC to carcinogenesis. The search space is limited by the maximum clause-length of 5 literals and maximum variable depth 2.

5 Related Work

Besides the direct inspiration by the findings due to Gomes et al., this work is also related to the research of the *phase transition* effect. Phase transition has been observed in algorithms for solving difficult computa-

tional problems, namely NP-complete ones such as the constraint satisfaction problem (CSP) [13]. A *constraint tightness* parameter $p \in < 0; 1 >$ can be calculated for any CSP instance. According to empirical studies [13], the expected time to solve a CSP is small for values of p close to 0 (phase of 'many solutions available') or 1 (phase of 'inspecting a small search tree') and grows dramatically for p close to a *critical value* p_{cr} (transition between the two phases). In the surrounding of p_{cr}, the costs of solving CSP instances not only show a high mean, but also a large variability. Frost el al. [2] approximate the cost distributions of instances with p close to p_{cr} with various closed-form distributions. They point out (independently of Gomes el al.) the long tails of these distributions and report that "problems that are not solved early are likely to take a long time". The fundamental bridge between such findings and ILP is the fact that the first-order subsumption problem can be mapped onto a CSP [7]. Botta et al. then show [8] that a typical ILP program (FOIL) tends to "induce hypotheses generating matching problems located inside the phase transition region"; Giordana and Saitta report a similar observation [3] in real-world domains, including mutagenesis. Combining the referred results, the heavy-tailed effect had been expectable before our study, which can thus be seen as an empirical verification of this implicit expectation. Unlike the previous studies where statistics were measured for a collection of the proving (subsumption check) problem instances, we measured distributions on a collection of complete hypothesis-searching cycles, each containing a number of subsumption tests.

The way statistical observations are exploited towards efficiency improvements also distinguishes our approach from the mentioned related work. Sebag and Rouveirol [11] apply a stochastic algorithm in order to accelerate the subsumption-test and Giordana and Saitta [3] develop an on-line complexity estimator which can potentially be used for the same purpose. Our approach, on the other hand, allows to adopt the RRR technique to reduce the complexity of the hypothesis search in its entirety.

As far as the randomized technique of traversing through the subsumption lattice is concerned, to our best knowledge, there is only indirectly related work to our study. Serra et al [12] show that starting the search for a hypothesis from random seed formulas, instead than top-down, can be beneficial. Randomized search in an ILP system has been assessed in [14].[4]

[4] We are also aware of the talk of Stephen Muggleton at the Machine Intelligence workshop in 2001 about randomization techniques in Progol but as we gather, there is no written account on that talk.

6 Conclusions

Our study has shown that the phenomenon of heavy-tailed runtime distributions occurs in two important experimental domains of ILP and we believe that it is typical to many other domains. Testing this hypothesis is a part of our future work.

This observation lead to the utilization of the technique of randomized rapid restarts which we reformulated for sakes of ILP. To apply this method, the exhaustive lattice search was randomized in such a way that we selected randomly the clause where the search was started. Randomized rapid restarts may then be used to reduce the average time required to find a clause with desired properties. A natural question is whether reducing the average runtime of the search procedure randomized in this way may lead to outperforming the deterministic top-down or bottom-up search. But clearly, if we do not impose a prior probability distribution on clauses (or e.g. on clause-lengths), there is no reason to expect that a search starting from the most general (most specific) element will be systematically faster than the average search starting in a random element of the lattice.

Using the technique of randomized rapid restarts, we were able to significantly reduce the search times in large hypothesis spaces of both of the tested domains.

7 Acknowledgements

We thank the ILP'02 referees for pointing us to some very relevant articles. Also, the ILP'02 audience contributed much to the paper by motivating us to relate our study to the phase transition research. Filip Zelezny greatly acknowledges the support from the EU project INCO 977102 ILPnet2 and the Czech Technical University grant CTU 0209013. David Page was supported by the U.S. National Science Foundation grant 9987841 and a U.S. DARPA EELD grant.

References

1. L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*. Wadsworth, Belmont, 1984.
2. Daniel Frost, Irina Rish, and Lluis Vila. Summarizing CSP hardness with continuous probability distributions. In *AAAI/IAAI*, pages 327–333, 1997.
3. A. Giordana and L. Saitta. Phase transitions in relational learning. *Machine Learning*, 2000.

4. C. P. Gomes, B. Selman, N. Crato, and H. A. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24(1/2):67–100, 2000.
5. C. P. Gomes, B. Selman, and H. A. Kautz. Boosting combinatorial search through randomization. In *AAAI/IAAI*, pages 431–437, 1998.
6. http://web.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph.html.
7. J. Maloberti and M. Sebag. Theta-subsumption in a constraint satisfaction perspective. volume 2157 of *Lecture Notes in Artificial Intelligence*, pages 164–178. Springer-Verlag, September 2001.
8. M.Botta, A.Giordana, L.Saitta, and M.Sebag. Relational learning: hard problems and phase transitions. In *6th Congress of the Italian Association for Artificial Intelligence*. Springer-Verlag, 1999.
9. S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.
10. S. Muggleton and L. De Raedt. Inductive logic programming: Theory and methods. *Journal of Logic Programming*, 19/20:629–679, 1994.
11. Michele Sebag and Celine Rouveirol. Resource-bounded relational reasoning: Induction and deduction through stochastic matching. *Machine Learning*, 38(1/2):41–62, January 2000.
12. A. Serra, A. Giordana, and L. Saitta. Learning on the phase transition edge. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 921–926. Morgan Kaufmann, 2001.
13. Barbara M. Smith and Martin E. Dyer. Locating the phase transition in binary constraint satisfaction problems. *Artificial Intelligence*, 81(1-2):155–181, 1996.
14. A. Srinivasan. A study of two probabilistic methods for searching large spaces with ILP. Technical Report PRG-TR-16-00, Oxford University Computing Laboratory, Oxford, 2000.
15. A. Srinivasan and R. King. Carcinogenesis predictions using ilp. In S. Džeroski and N. Lavrač, editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297 of *Lecture Notes in Artificial Intelligence*, pages 3–16. Springer-Verlag, 1997.
16. A. Srinivasan and R.D. King. Feature construction with Inductive Logic Programming: a study of quantitative predictions of biological activity aided by structural attributes. *Data Mining and Knowledge Discovery*, 3(1):37–57, 1999.
17. A. Srinivasan, S. Muggleton, M.J.E. Sternberg, and R.D. King. Theories for mutagenicity: A study in first-order and feature-based induction. *Artificial Intelligence*, 85(1,2), 1996.
18. R. Walpole and R. Myers. *Probability and Statistics for Engineers and Scientists*. Collier Macmillan, New York, 1978.