A Monte Carlo Study of Randomised Restarted Search in ILP

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

[1] Dept. of Cybernetics
School of Electrical Engineering
Czech Institute of Technology (ČVUT) in Prague
Karlovo Nám. 13, 121 35 Prague, Czech Republic
zelezny@fel.cvut.cz
[2] IBM India Research Laboratory
Block 1, Indian Institute of Technology
New Delhi 110 016, India
ashwin.srinivasan@in.ibm.com
[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 statistical performance surveys of search algorithms in difficult combinatorial problems have demonstrated the benefits of randomising and restarting the search procedure. Specifically, it has been found that if the search cost distribution (SCD) of the non-restarted randomised search exhibits a slower-than-exponential decay (that is, a "heavy tail"), restarts can reduce the search cost expectation. Recently, this heavy tail phenomenon was observed in the SCD's of benchmark ILP problems. Following on this work, we report on an empirical study of randomised restarted search in ILP. Our experiments, conducted over a cluster of a few hundred computers, provide an extensive statistical performance sample of five search algorithms operating on two principally different ILP problems (artificially generated graph data and the well-known "mutagenesis" problem). The sample allows us to (1) estimate the conditional expected value of the search cost (measured by the total number of clauses explored) given the minimum clause score required and a "cutoff" value (the number of clauses examined before the search is restarted); and (2) compare the performance of randomised restarted search strategies to a deterministic non-restarted search. Our findings indicate that the cutoff value is significantly more important than the choice of (a) the specific refinement strategy; (b) the starting element of the search; and (c) the specific data domain. We find that the optimal value for the cutoff parameter remains roughly stable across variations of these three factors and that the mean search cost using this value in a randomised restarted search is up to three orders of magnitude (i.e. 1000 times) lower than that obtained with a deterministic non-restarted search.

1 Introduction

Computer programs now collectively termed "predictive Inductive Logic Programming" (predictive ILP) systems use domain-specific background information and pre-classified sample data to construct a set of first-order rules for predicting the classification labels of new data. Despite considerable diversity in the applications of ILP, (see [3] for an overview) successful implementations have been relatively uniform, namely, engines that repeatedly examine sets of candidate rules to find the "best" ones. Any one step of this sequence is an enumerative search—usually some approximation to the optimal branch-and-bound algorithm—through a space of possible rules. This choice of search method can critically affect the performance of an ILP system on non-trivial problems. Enumerative search methods, despite their attractive simplicity, are not *robust* in the sense of achieving a balance between efficiency and efficacy across different problems [4]. For many practical problems that engender very large spaces of discrete elements, enumerative search, however clever, becomes intractable and we are forced to take seriously Trefethen's Maxim No. 30 [1]: "If the state space is huge, the only reasonable way to explore it is at random."

Recently, research into the development of efficient automatic model-checkers has led to the development of novel randomised search methods that abandon optimality in favour of "good" solutions. Prominent examples are the GSAT and WalkSat methods checking the satisfiability of propositional formulae [9], as randomised alternatives to the (enumerative) Davis-Putnam solver. In conjunction with this, there is now a vigorous line of research that investigates properties of large search spaces corresponding to difficult combinatorial problems [5]. Some intriguing properties have been identified, such as the high irregularity of the search spaces and "heavy-tailedness" of the cost distributions of search algorithms used. Such properties manifest themselves in a large collection of real-world problems and have been the inspiration for the design of randomised restarted search procedures. The basic idea of these procedures is simple: if each search trial has a small, but fixed probability of finding a good clause, then the probability of finding a good clause in a sequence of such trials can be made quite high very rapidly. Put differently, the SCD from the sequence has an exponential decay.

Previously, the heavy-tailed character of search cost distributions was reported in the context of the first-order rule search conducted in ILP [11]. There, a simple adaptation of a method known as Randomised Rapid Restarts [6] was shown to result in a considerable reduction of clause search cost. Here, we extend that investigation as follows:

1. We adapt a family of randomised restarted search strategies into an ILP system and present all of them as instantiations of a general algorithm.
2. We design and conduct an extensive Monte Carlo study that allows us to model the statistical relationships between the search cost, the score of the best clause and the number of clauses explored in each restart (called the "cutoff" value in the search algorithm).

Our experiments are conducted with data drawn from two domains: artificially generated, noise-free, graph problems, in which "target" theories can be modelled by a single, long clause (up to 10 literals in the body of the clause); and the well-known mutagenesis problem, which is typically modelled by multiple, relatively short clauses (typically up to 5 body literals). Although the natures of the problems are quite different to each other, the main statistical findings relate equally to both the domains.

The paper is organised as follows. In the next section we describe the clause search strategies considered and the performance metric used to evaluate the strategies. Details of the Monte Carlo study of these strategies and the dependence of their performance on some important parameters is in Section 3, where we also discuss our results and formulate questions requiring further investigation. Section 4 concludes the paper.

2 Search

We are principally concerned with performing a search in the clause subsumption lattice bounded at one end by a finite most specific ("bottom") clause derived using definitions in the background knowledge, a depth-bounded mode language, and a single positive example (the "saturant": see [8] for more details on the construction of this clause). For simplicity, we will assume the specification of the depth-bounded mode language to be part of the background knowledge.

2.1 Strategies

The five search strategies that we investigate in this paper are: (1) A deterministic general-to-specific search (DTD); (2) A randomised general-to-specific search (RTD); (3) A rapid random restart search (RRR); (4) A randomised search using the GSAT algorithm (GSAT); and (5) A randomised search using the WalkSAT algorithm (WSAT). All five strategies can be viewed as variations of a general search procedure shown in Fig. 1. Differences between the individual strategies arise from the implementation of the commands in bold-face (summarised in Table 1). All strategies include restarts (if γ is a finite value). Restarting DTD clearly results in simply repeating the search.

As further clarification of the entries in Table 1, we note the following:

Saturant selection. A deterministic implementation ('D') of the first **Select** command (Step 3 in Fig. 1) results in the first positive example in the presented example sequence is chosen as the saturant. A randomised implementation ('R') results all examples having a uniform probability of selection.

Start clause selection. A deterministic implementation ('D') of the second **Select** command (Step 4), results in the search commencing with the the most general definite clause allowable. A randomised implementation ('R') results in a clause selected with uniform probability from the set of allowable clauses (see [11] for more details on how this is achieved).

search$(B, H, E, s^{suf}, c^{all}, \gamma)$: Given background knowledge B; a set of clauses H; a training sequence $E = E^+, E^-$ (i.e. positive and negative examples); a sufficient clause score s^{suf} $(-\infty \leq s^{suf} \leq \infty)$; the maximum number of clauses the algorithm can evaluate c^{all}, $(0 < c^{all} < \infty)$; and the maximum number of clauses evaluated on any single retart or the 'cutoff' value γ $(0 < \gamma \leq \infty)$, returns a clause D such that $B \cup H \cup \{D\}$ entails at least one element e of E^+. If fewer than c^{all} clauses are evaluated in the search, then the score of D is at least s^{suf}.

1. $S := -\infty$; $C := 0$; $N := 0$
2. repeat
3. **Select** e^{sat} from E^+
4. **Select** D_0 such that $D_0 \succeq_\theta \bot(e^{sat}, B)$
5. $Active = \emptyset$; $Ref = \{D_0\}$
6. repeat
7. $S^* = \max_{D_i \in Ref} \underline{eval}_{B,H}(D_i)$; $D^* := \arg\max_{D_i \in Ref} \underline{eval}_{B,H}(D_i)$
8. if $S^* > S$ then $S := S^*$; $D := D^*$
9. $N := N + |Ref|$
10. $Active :=$ **UpdateActiveList**$(Active, Ref)$
11. $Prune :=$ **Prune**$(Active, S^*)$
12. $Active := Active \setminus Prune$
13. **Select** D^{curr} from $Active$; $Active := Active \setminus D^{curr}$
14. $Ref :=$ **Refine**$_{B,H,(\gamma-N)}(D^{curr})$
15. until $S \geq s^{suf}$ or $C + N \geq c^{all}$ or $N = \gamma$
16. $C := C + N$; $N := 0$
17. until $S \geq s^{suf}$ or $C \geq c^{all}$
18. if $S = -\infty$ then return e^{sat} else return D^*.

Fig. 1. A general skeleton of a search procedure—possibly randomised and/or restarted—in the clause subsumption lattice bounded by the clause $\bot(e^{sat}, B)$. This clause is derived using the saturant e^{sat} and the background knowledge B. In Step 4, $\succeq_\theta$ denotes Plotkin's (theta) subsumption between a pair of Horn clauses. Individual strategies considered in this paper are obtained by different implementations of the bold-typed commands. Clauses are scored by a finite evaluation function $\underline{eval}$. Although in the formal notation in Step 7 the function appears twice, it is assumed that the 'max' and 'arg max' operators are computed simultaneously. In Step 11 **Prune** returns all elements of *Active* that cannot possibly be refined to have a better score than S^*. If the number of refinements of the current clause is greater than $(\gamma - N)$, **Refine** returns only the first $(\gamma - N)$ computed refinements, to guarantee that no more than γ clauses are evaluated between restarts. The search is terminated when score s^{suf} is reached or c^{all} clauses have been evaluated, and restarted (from Step 3) when γ clauses have been evaluated since the last restart. If all **Select** commands are deterministic then restarting (setting $\gamma < c^{all}$) results in mere repetitions of the identical search.

Strategy → ↓ *Step*	DTD	RTD	RRR	GSAT	WSAT
Saturant selection (**Select** in Step 3)	D	R	R	R	R
Start clause selection (**Select** in Step 4)	D	D	R	R	R
Update active list (**UpdateActiveList** in Step 10)	C	C	C	G	G
Next clause selection (**Select** in Step 13)	D	R	D	D	R
Pruning (**Prune** in Step 11)	Y	Y	N	N	N
Refinement (**Refine** in Step 14)	U	U	B	B	B

Table 1. Implementation differences amongst the different search strategies. The entries are as follows: 'D' stands for 'deterministic', 'R' for 'randomised', 'G' for greedy, 'C' for complete, 'Y' to denote that pruning occurs, 'N' that pruning does not occur, 'U' for uni-directional refinement (specialisation only) and 'B' for to bi-directional refinement (specialisation and generalisation). See text for more details on these entries.

Update active list. A greedy implemention ('G') of the **UpdateActiveList** function (Step 10) results in the active list containing only the newly explored nodes (elements of the Ref). A complete implementation ('C') results in *Active* containing all elements (including elements of Ref).

Next clause selection. A deterministic implementation ('D') of the last **Select** command (Step 13) results in the clause with the highest score being chosen from the *Active* list (with ties being decided by the appearance order of clauses). A randomised implementation ('R') results in a random choice governed by the following prescription:

- With probability 0.5, select the clause with the highest score in the *Active* list.
- Otherwise, select a random clause in the *Active* list with probability proportional to its score.

Pruning. 'Y' denotes that pruning is performed, which results in a possibly non-empty set being returned by the **Prune** command (Step 11). A 'N' implementation means that an empty set is returned.

Refinement. The 'U' implementation of **Refine** command (Step 14) results in refinements that are guaranteed to be specialisations of the clause being refined. The 'B' implementation produces the (most general) specializations and (most specific) generalizations of the refined clause.

2.2 Evaluation

Informally, given some clause evaluation function, for each search strategy we ask:

How many clauses must be searched to achieve a desired clause score?

Here, we treat the number of clauses searched as representative of the 'search cost' and quantify this cost-score trade-off by the the expected value of the

smallest cost needed to achieve or exceed a desired score s^{suf} [1], given an upper bound γ on the clauses searched on any single restart. Thus, for each strategy we wish to estimate:

$$cost(s^{suf}) \equiv E[C|s^{suf}, \gamma] \tag{1}$$

The following points are evident, but worth restating:

1. Let us assume that strategy St_1 is found to achieve, on average, a desired score s^{suf} significantly faster than strategy St_2. Strictly speaking, even if the clauses added successively to a constructed theory do not reference each other, we cannot conclude that a set-covering algorithm employing St_1 will be more efficient than that using St_2. This is because in the cover algorithm, the individual clause search procedures are not statistically independent events (since one influences the following by removing a subset of the positive examples).[2]
2. We are only concerned here with the search cost in finding a clause with a given score on the training set. This does not, of course, translate to any statement about the performance of the clause found on new (test) data. It is certainly interesting and feasible to also investigate whether and how the generalization rate is statistically dependent on the procedure used to arrive at an acceptable clause, given a required score. This is, however, outside the scope of this study.
3. A search cost of immediate interest is the processor time occupied by a strategy. By adopting instead to measure the number of clauses searched, we are unable to quantify precisely the exact time taken by each strategy. Besides the obvious hardware dependence, research elsewhere [2] has shown that the cost of evaluating a clause can vary significantly depending on the nature of the problem addressed and formulation of the background knowledge. In this study we are concerned with obtaining some domain-independent insight into the five strategies.
4. Our evaluation of the strategies may have well been based on a different question, namely:

   What clause score is achieved given an allocated search cost?

   Here we would consider the expected score given an allocated cost c^{all}, that is $score(c^{all}) \equiv E[S|c^{all}, \gamma]$. Although the sample resulting from the Monte Carlo study can be used to evaluate the strategies in this way, space requirements confine this study to the former question, which we believe is of more immediate interest to practitioners of ILP.

[1] At any stage of the search, the score value maintains the highest clause evaluation so far obtained in the search. In other words, within a particular search execution, the score value is a non-decreasing function of the cost (i.e. the number of clauses searched).

[2] The conclusion would however be correct for many other ruleset induction algorithms where the events are independent, such as CN2-like unordered rulesets, various other voting rulesets etc.

3 Empirical Evaluation

3.1 Materials

Data Experiments were conducted using two ILP benchmarks. The first data set describes a set of 5,000 directed graphs (containing in total 16,000 edges). Every node in a graph is coloured to red or black. Each graph is labelled positive if and only if it contains a specific (coloured) subgraph which can be represented by a predefined clause of 10 body literals. Although this clause can be viewed as the target theory, there exist other clauses (subgraphs) in the search lattice that precisely separate the positive examples from the negatives. The second problem – mutagenesis prediction – has been discussed extensively in ILP literature and we use here one of the datasets described in our previous publication, namely the data pertaining to 188 "regression-friendly" compounds [10]. Table 2 describes the principal differences between the two data sets. The graph data set is available on request to the first author and the software for its generation can be obtained from the third author. The mutagenesis dataset can be obtained via anonymous ftp to `ftp.comlab.ox.ac.uk` in the directories `pub/Packages/ILP/Datasets/mutagenesis/aleph`.

Property	**Graphs**	**Mutagenesis**
Origin	Artificially generated	Real-life
Noise	No	Yes
'Target' theory	Yes	No
'Good' theory	One long clause (10 lits)	Several short clauses (up to 5 body lits)
# pos/neg examples	20/20	125/63

Table 2. Differences between the experimental data sets.

Algorithm and Machines All experiments use the ILP program Aleph. Aleph is available at: `http://www.comlab.ox.ac.uk/oucl/research/areas/machlearn/Aleph/aleph.pl`. Additional code implemented to Aleph for purposes of the empirical data collection can be obtained from the first author. The computation was distributed over the Condor computer cluster at the University of Wisconsin, Madison. All subsequent statistical analysis of the collected data was done by means of the R statistical package. The R procedures developed for this purpose can be obtained from the first author.

3.2 Method

Recall that we are interested in estimating the conditional expected cost value $E[C|s^{suf}, \gamma]$ for each of the five strategies in Section 2.1. A straightforward way

to collect the required statistical sample needed to estimate the expected value for a given search strategy would thus be to run a number of instances of the algorithm in Fig. 1, each with a different setting of the sufficient score parameter s^{suf} and the restart cutoff parameter γ, each time recording the resulting value of C. This approach would however perform a lot of redundant computation. Instead we adopt the following method:

For each problem (5,000 graphs and 188 mutagenic chemicals)
For each randomized strategy (RTD, RRR, GSAT, WSAT)

1. $\gamma = \infty$, $c^{all} = c_{max}$ (some large value: see notes below), $s^{suf} = s_{max}$ (the maximum possible clause score: see notes below)
2. for $i = 1$ to $\#Runs$
   (a) Call $search(B, \emptyset, E, s^{suf}, c^{all}, \gamma)$ (see Fig. 1).
   (b) Record the 'performance' vector $c_i = [c_i(0), \ldots, c_i(s_{max_i})]$ where $c_i(s)$ is the number of clauses evaluated before achieving (or exceeding) score s for the first time and s_{max_i} is the maximum score achieved on run i.
3. Compute the expected cost from the performance vectors recorded.

The following details are relevant:

1. The method assumes a finite, integer-valued scoring function. In the experiments we evaluate the score of a clause D as $P(D) - N(D)$ where $P(D)$ and $N(D)$ are the numbers of positive and negative examples 'covered' by D. That is, given positive and negative examples E^+, E^- let $E_p \subseteq E^+$ s.t. $B \cup \{D\} \models E_p$ and $E_n \subseteq E^-$ s.t. for all $e_n \in E_n$, $B \cup \{D\} \cup \{e_n\} \models \square$. Then $P(D) = |E_p|$ and $N(D) = |E_n|$. Rejecting all clauses for which $P(D) < N(D)$, the range of the score is $0 \ldots P$ where $P = |E^+|$. Thus in Step 1 $s_max = P$.
2. In these experiments c_{max} was set to 200,000 and $\#Runs$ was set to 6,000. Thus, the empirical sample after execution of Step 2 consists of 6,000 performance vectors. Each performance vector has at most $P = |E^+|$ elements (fewer elements are possible if score P was not achieved on a run).
3. Step 3 requires the computation of expected cost (i.e. the number of evaluated clauses) for any value of γ and s^{suf}. In Appendix A we describe how the sample of 6,000 performance vectors can be used to obtain an unbiased estimate this value.

The method above does not refer to the non-restarted strategy DTD. DTD is deterministic and thus a single run (and corresponding single performance vector) should be sufficient to describe its performance. However, unlike the other strategies, DTD does not select the saturated example randomly, but it selects the first positive example for saturation. To avoid artifacts arising from any particular example ordering, we obtain instead an average conditional cost. That is, we thus perform Step 2a above $P = |E^+|$ times, each time selecting a different saturant. This results in P performance vectors: Appendix A shows how these performance vectors can be used to obtain a biased (lower bound) estimate of the expected cost for any value of s^{suf}.

3.3 Results

Figures 2–5 show diagrammatically the estimated expected number of evaluated clauses ('expected cost value') as a function of the restart cutoff parameter γ. Each graph has 2 kinds of plots: horizontal lines, representing (a lower bound on) the cost of the deterministic strategy DTD; and non-constant lines representing a randomised strategy. Each line is tagged with a number, which represents the sufficient clause score s^{suf}. Thus, in Fig. 2, for the plot labelled "Mutagenesis: RTD", the horizontal line tagged "10" with a constant cost of 10,000 indicates that the expected number of clauses explored by DTD before finding a clause with score 10 is 10,000. The corresponding costs, for different values of the cutoff parameter γ for RTD is shown by the lowermost non-constant line (each entry is tagged by "10"). Figures associated to the graphs domain contain 20 such plots (corresponding to all possible clause scores: some plots may overlay), figures belonging to the mutagenesis domain contain 7 plots (corresponding to $s^{suf} = 10, 20, \ldots 70$).[3] In all the diagrams, the highest vertical (*cost*) coordinate should be interpreted as: "c_{max} or higher" as all points corresponding to an expected cost in the interval $[c_{max}, \infty]$ are plotted with the vertical coordinate c_{max}.

Broadly, there are remarkable similarities in the plots from different strategies for a given problem domain, as well as from the same strategy for different problem domains. The principal trends are these:

1. The setting of the cutoff parameter γ has a very strong impact on the expected cost. A choice close to its optimal value may reduce the expected cost over DTD up to a 1,000 times for mutagenesis with RRR, GSAT or WSAT[4]. With RTD, the reduction is even higher for very low s^{suf}.
2. $\gamma \approx 100$ is a 'good' choice for all the investigated strategies in both domains. For the graph problems, the value is close to optimal (in the considered range of γ) for all strategies and for all of them it leads to similar expected costs, with the exception of WSAT where the costs are significantly higher[5] In mutagenesis, $\gamma \approx 100$ is a local minimum for all strategies (for RTD it is even the optimum) when $s^{suf} > 30$.
3. $\gamma < 10$ appears to be uniformly a 'bad' choice for all randomised strategies on the graph problems.
4. For a given domain, expected costs of the different restarted strategies are roughly similar, especially in the vicinity of $\gamma = 100$ (once again, WSAT on the graphs problem is the exception).

[3] We do not plot lines for further scores $s^{suf} = 80, 90 \ldots P$ since these were not achieved by any of the strategies in any of the runs.

[4] In mutagenesis, the high expected costs of DTD are due to several positive examples, whose saturation leads to an unfavorable search space.

[5] A tentative explanation of this may lie in the fact that the explored clause score is a particularly good heuristic for node selection in the noise-free graph problems. In these circumstances the randomization inherent in WSAT may be having a detrimental effect on the mean performance of WSAT.

5. For both domains, other than very high values of s^{suf}, the costs of the restarted strategies are uniformly lower than that of the non-restarted strategy DTD for a wide range of γ ($100 \leq \gamma \leq 10,000$).

3.4 Discussion

For large intervals of the cutoff parameter γ, the restarted randomised search strategies (RTD, RRR, GSAT, WSAT) exhibit similar performance, outperforming the non-restarted deterministic clause search (DTD) in terms of the cost. However, two issues require further investigation before conclusions can be made concerning the ranking of the strategies. First, DTD (and RTD) always begins the search with the most general definite clause allowed by the language restriction. It is therefore biased towards evaluating shorter clauses than RRR, GSAT or WSAT which often means spending less total evaluation time for the same number of clauses scored. If processor time is assigned to the search cost, it is possible that both top-down strategies may improve their relative performance with respect to the restarted methods (of course, the empirical results suggest that the latter may evaluate far fewer clauses). Second, our measurement scheme does not assign any cost to the computation needed to construct a bottom clause for each restart. Clearly, the corresponding overhead time grows linearly with decreasing γ (there will be more restarts). As a result, when computation time is viewed as the search cost, the optimum value of γ is likely to shift to a higher value than that determined in our experiments, and the large performance superiority of the the restarted methods observed for small γ is likely to reduce.

It is encouraging that two apparently very different domains and all restarted strategies have yielded a similar range of 'good' values for the γ parameter. However, the plots, especially on in the graphs domain, highlight a further aspect: there is quite a sharp increase in costs for values of γ that are just below the optimum. This suggests that it would thus be useful to consider a restarted algorithm that is less dependent on the location of the optimal γ. A solution may lie in a cutoff value gradually (for example, geometrically) growing with each restart. This idea of *dynamic* restarts has been considered before [7] and may result in a more robust search algorithm.

4 Concluding Remarks

Search is at the heart of all modern Inductive Logic Programming systems, and most have thus far employed well-known deterministic search methods. In other fields confronted with very large search spaces, there is now substantial evidence that the use of randomised restarted strategies yield far superior results to deterministic ones (often making the difference between getting a good solution, or none at all). Unfavourable conditions for deterministic search observed in those fields—the heavy-tailed character of search cost distributions—have also been reported in the context of the search conducted by many ILP systems [11]. In

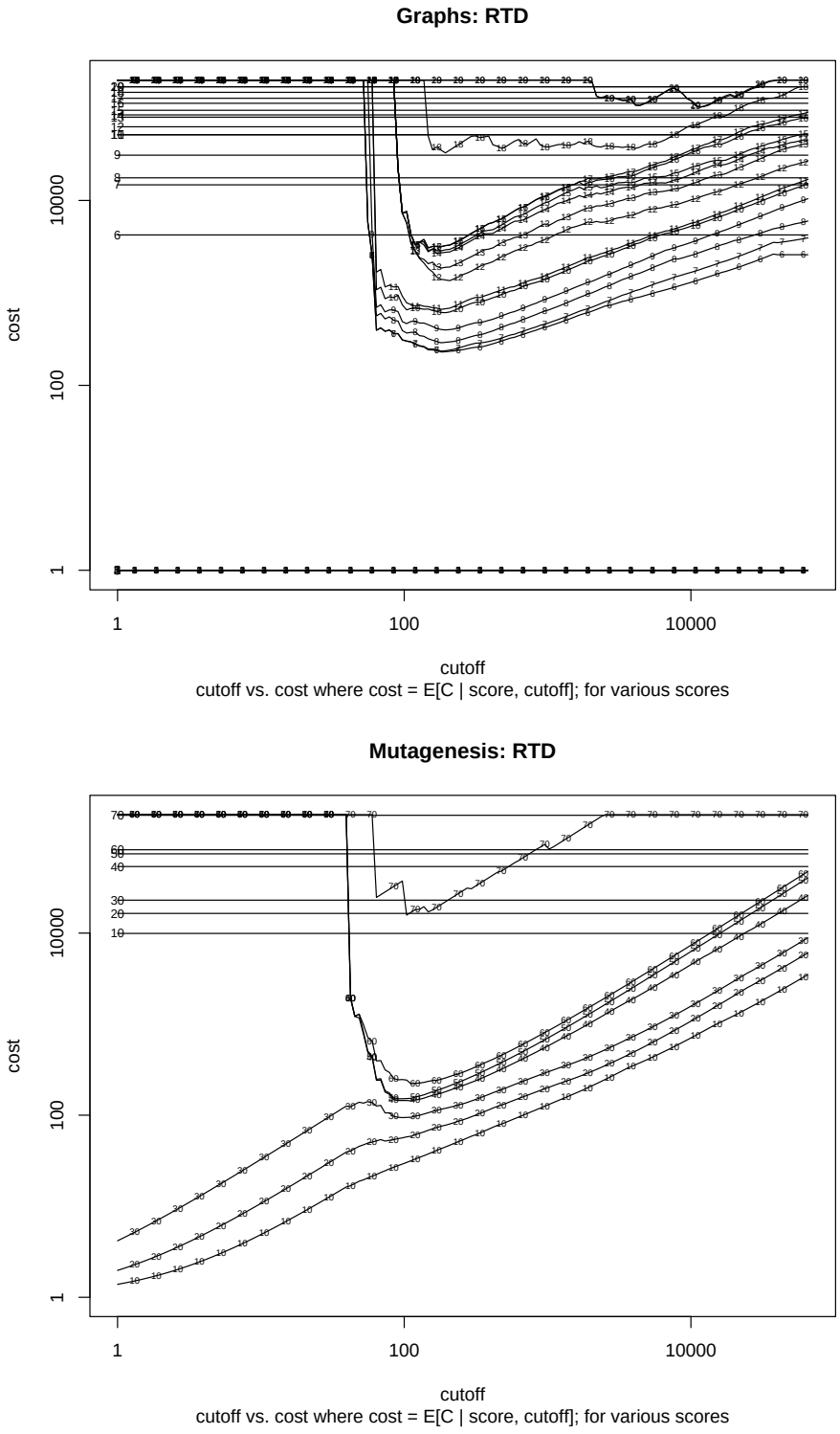

Fig. 2. Randomised Top-Down vs. Deterministic Top-Down (horizontal lines) search

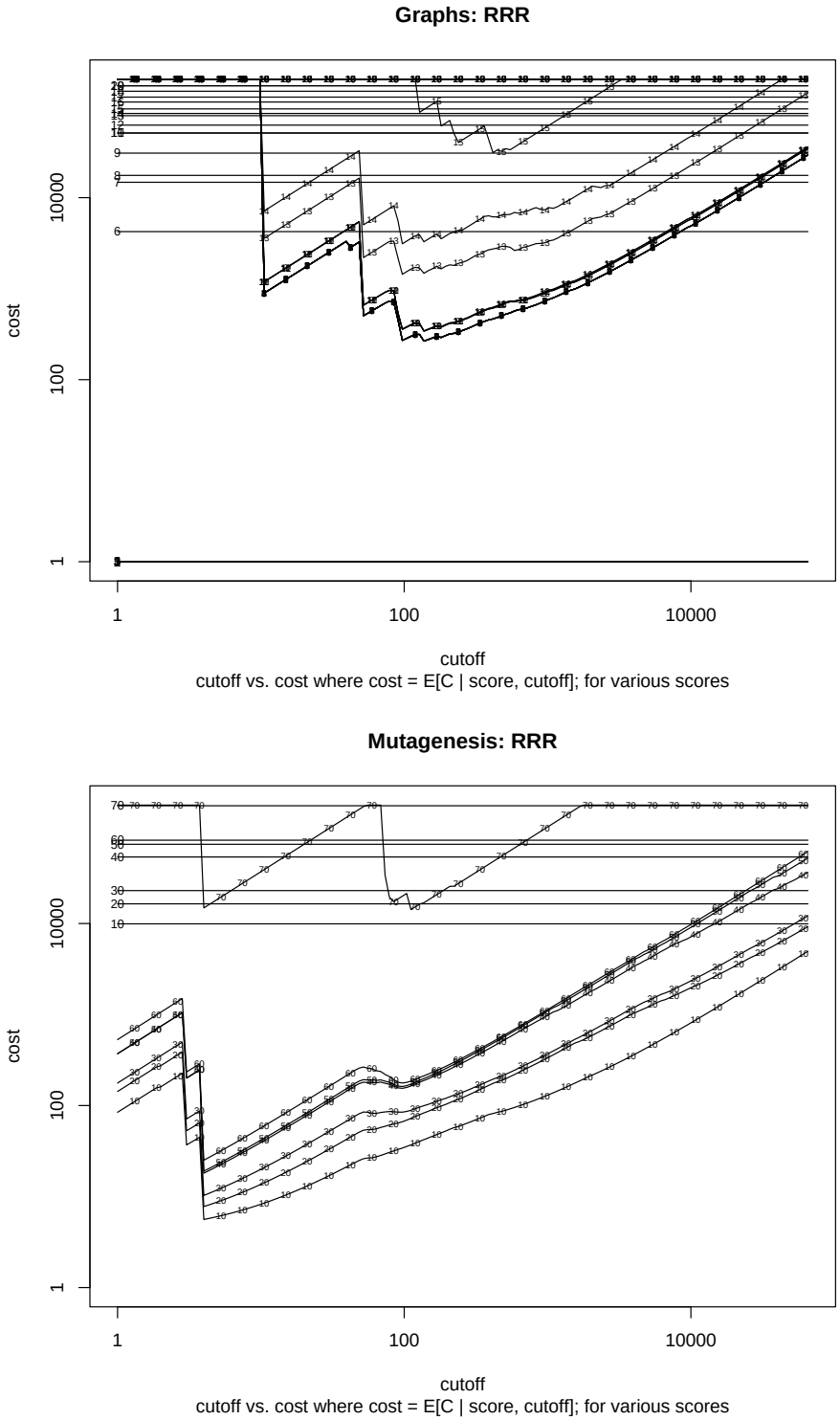

Fig. 3. 'Randomised Rapid Restarts' vs. Deterministic Top-Down (horizontal lines) search

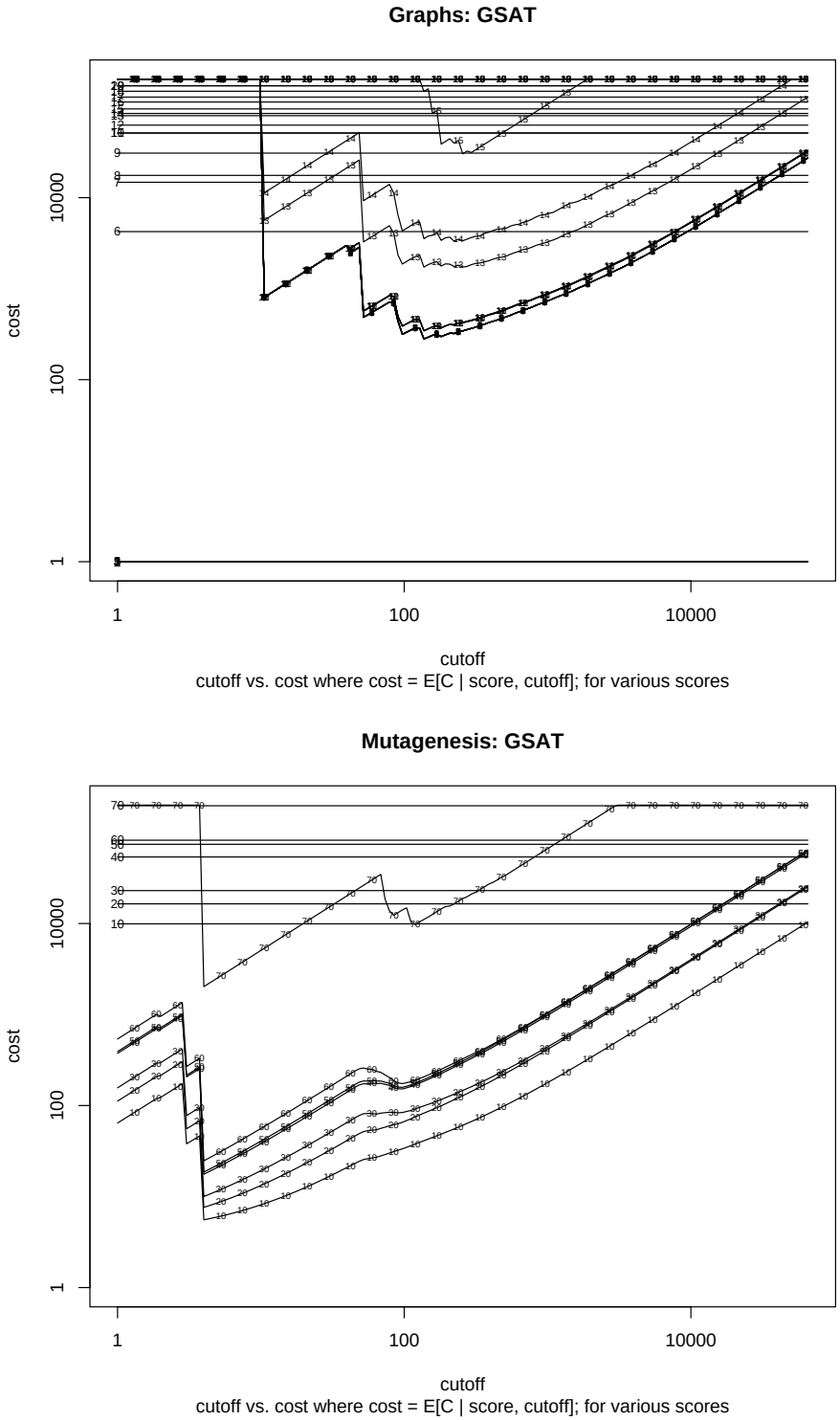

Fig. 4. GSAT vs. Deterministic Top-Down (horizontal lines) search

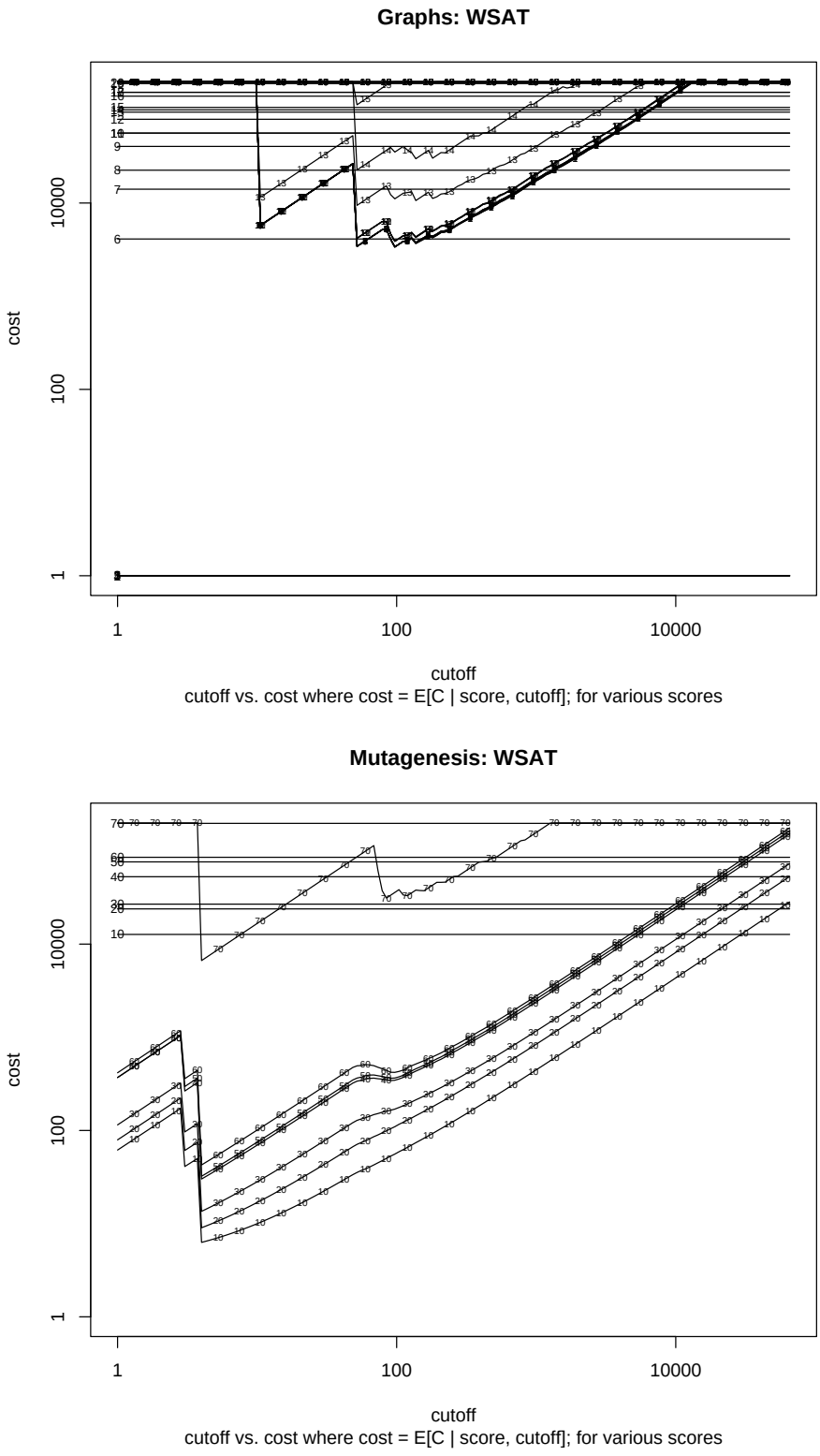

Fig. 5. WSAT vs. Deterministic Top-Down (horizontal lines) search

this paper, we have presented what appears to be the first systematic study of a number of randomised restarted search strategies for ILP. Specifically, we have adopted a Monte Carlo method to estimate the search cost—measured by the number of clauses explored before a 'good' clause is found—of these strategies on two quite different ILP problems. The result is encouraging: in each domain, for a wide range of values for a parameter γ controlling the number of restarts, randomised restarted methods have a lower search cost than a deterministic general-to-specific search.

The performance sample generated has also provided some useful insights into the randomised techniques. First, it appears that there may a 'good' value for γ that works adequately for across many domains. Second, although they differ in the choice of the first element of the search and the refinement strategy employed, all randomised methods appear to perform similarly. Third, there may be some value in exploring a randomised restarted search strategy with a dynamically growing value of γ.

While accepting all the usual caveats that accompany an empirical study such as this, we believe the results here to be sufficiently encouraging for researchers to explore further the use of randomised restarted search methods in ILP, especially with other data domains and scoring functions.

A Calculating Expected Cost from Performance Vectors

We describe here a technique for estimating the conditional expected cost value $E[C|s^{suf}, \gamma]$ for each of the five strategies in Section 2.1. We exploit the fact that the expected cost for a strategy with arbitrarily set s^{suf} and γ parameters can be estimated from a sample of executions of the algorithm in Fig. 1 where $s^{suf} = P$ (where P is the maximum possible score) and $\gamma = \infty$. Since we require all trials terminate in a finite time, we let c^{all} equal to some large finite value c_{max} (for the experiments in the paper $c_{max} = 200{,}000$) for each of the random trials. As we shall see below, setting a finite c^{all} will bias the estimates for non-restarted strategies, but will still allow us to obtain unbiased estimates for restarted strategies for all values of $\gamma < c^{all}$.

Recall that executing the experimental method described in Section 3.2 results, for each strategy and problem, in a set of 'performance' vectors $c_i = [c_i(0), \ldots, c_i(s_{max_i})]$ where $1 \leq i \leq 6000$ for the randomised strategies RTD, RRR, GSAT and WSAT; and $1 \leq i \leq P = |E^+|$ for the deterministic strategy DTD. With each c_i $c_i(s)$ is the number of clauses evaluated before achieving (or exceeding) score s for the first time and s_{max_i} is the maximum score achieved on run i

For DTD, $E[C|s^{suf}, \gamma = \infty]$ is obtained by simply averaging the $c_i(s^{suf})$ over all i's. However, it is possible that in some of the trials i, the maximum score P is not achieved after evaluating c_{max} clauses. In such cases, there exist values $s^{suf} \leq P$ such that $c_i(s^{suf})$ is not defined. Here we set $c_i(s^{suf}) \equiv c_{max}+1$. Thus, the cost we associate to non-restarted strategies will represent *lower bounds* of their expected cost.

For the restarted searches RTD, RRR, GSAT and WSAT, let the sequence of steps 4–14 in Fig. 1 be called a *try*. The probability that s^{suf} is achieved in the t-th try (and not in tries $1 \dots t-1$), given the cutoff value γ, is

$$F(\gamma|s^{suf}) \left(1 - F_s(\gamma|s^{suf})\right)^{t-1} \tag{2}$$

where the conditional cumulative distribution $F(x|s) = P(C \leq x|s)$ represents the probability of achieving or exceeding the score s having evaluated x of fewer clauses. It is estimated for a given score s from the empirical data as the fraction

$$F(x|s) \approx \frac{|\{c_i|c_i(s) \leq x\}|}{|\{c_i\}|} \tag{3}$$

Note that the consequence of s not being achieved in a particular run i is simply that the condition $c_i(s) \leq x$ does not hold. That is, run i is counted as a realization of the random trial with an 'unsuccessful' outcome. Thus to estimate $F(x|s)$ we do not need to assign a value to $c_i(s)$ in such a case (as was done above for DTD) and the the estimate remains unbiased.
The expected number of tries initiated before achieving s^{suf} is

$$E[T|s^{suf}, \gamma] = F(\gamma|s^{suf}) \sum_{t=1}^{\infty} t \left(1 - F(\gamma|s^{suf})\right)^{t-1} \tag{4}$$

It equals 1 for $F(\gamma|s^{suf}) = 1$ and for $F(\gamma|s^{suf}) = 0$ we set $E[T|s^{suf}, \gamma] = \infty$. If $0 < F(\gamma|s^{suf}) < 1$, it can be shown that Expression 4 converges to

$$E[T|s^{suf}, \gamma] = \frac{1}{F(\gamma|s^{suf})} \tag{5}$$

If we simply assumed that the algorithm evaluates exactly γ clauses in each try including the last, then the expected number of evaluated clauses would be

$$\bar{E}[C|s^{suf}, \gamma] = \gamma E[R|s^{suf}, \gamma] = \frac{\gamma}{F(\gamma|s^{suf})} \tag{6}$$

However, $\bar{E}[C|s^{suf}, \gamma]$ is imprecise because the algorithm may achieve s^{suf} evaluating fewer than γ clauses in the last try. The expected total number of clauses evaluated in all but the last try is

$$\gamma E[T-1|s^{suf}, \gamma] = \gamma \left(\frac{1}{F(\gamma|s^{suf})} - 1\right) \tag{7}$$

Due to the linearity of the expectation operator, we can determine the correct total expected cost by adding to the above value the expected number of clauses evaluated in the last try. For this purpose, consider the family of conditional probability distributions

$$D_t(n) = P(N = n|(t-1) * \gamma < C \leq t\gamma, s^{suf}, \gamma) \tag{8}$$

For $t = 1, 2, \ldots$, each D_t describes the probability distribution of the number of evaluated clauses in the t-th try under the specified parameters s^{suf}, γ, and *given* that the t-th try is the last in the search, ie. an acceptable clause is found therein. Since individual tries are mutually independent, the distributions D_t are identical for all t, that is, for an arbitrary t it holds $D_t(n) = D_1(n)$. Because in the first try it holds[6] that $N = C$, we can write

$$D_1(n) = P(C = n | C \leq \gamma, s^{suf}) \tag{9}$$

We did not include γ in the conditional part because its value does not affect the probability of the event $C = n$ given that $C \leq \gamma$, ie. given that no restart occurs. Applying basic probability algebra,

$$D_1(n) = \frac{P(C = n, C \leq \gamma | s^{suf})}{P(C \leq \gamma | s^{suf})} \tag{10}$$

If $n > \gamma$ then $D_1(n) = 0$. Otherwise, we can drop the $C \leq \gamma$ conjunct (implied by $C = n$) from the nominator expression:

$$D_1(n) = \frac{P(C = n | s^{suf})}{P(C \leq \gamma | s^{suf})} = \frac{F(n | s^{suf}) - F(n-1 | s^{suf})}{F(\gamma | s^{suf})} \tag{11}$$

Now we can calculate the expected number $E[N | (t-1) * \gamma < C \leq r\gamma, s^{suf}, \gamma]$ of clauses evaluated in the last try as

$$\sum_{n=1}^{\infty} n D_t(n) = \sum_{n=1}^{\gamma} n D_1(n) = \sum_{n=1}^{\gamma} n \frac{F(n | s^{suf}) - F(n-1 | s^{suf})}{F(\gamma | s^{suf})} \tag{12}$$

Summing up Eq. 7 with Eq. 12 we get the expected total number of evaluated clauses:

$$E[C | s^{suf}, \gamma] = \gamma \left(\frac{1}{F(\gamma | s^{suf})} - 1 \right) + \frac{\sum_{n=1}^{\gamma} n \left(F(n | s^{suf}) - F(n-1 | s^{suf}) \right)}{F(\gamma | s^{suf})} \tag{13}$$

Recall that the conditional distribution $F(.|.)$ used above can be estimated from the performance vectors as described by Eq. 3.

References

1. *http://web.comlab.ox.ac.uk/oucl/work/nick.trefethen/maxims.html.*
2. M. Botta, A. Giordana, L. Saitta, and M. Sebag. Relational learning as search in a critical region. *Journal of Machine Learning Research*, (4):431–463, 2003.
3. S. Dzeroski. Relational data mining applications: An overview. In *Relational Data Mining*, pages 339–364. Springer-Verlag, September 2001.

[6] Within the first try, the total number of evaluated clauses equals the number of clauses evaluated in the current try.

4. D. E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.
5. C. Gomes and B. Selman. On the fine structure of large search spaces. In *Proceedings the Eleventh International Conference on Tools with Artificial Intelligence ICTAI'99, Chicago, IL*, 1999.
6. 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.
7. H. Kautz, E. Horvitz, Y. Ruan, C. Gomes, and B. Selman. Dynamic restart policies, proceedings of the eighteenth.
8. S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.
9. B. Selman, H. J. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In Paul Rosenbloom and Peter Szolovits, editors, *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 440–446, Menlo Park, California, 1992. AAAI Press.
10. 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):277–299, 1996.
11. F. Železný, A. Srinivasan, and D. Page. Lattice-search runtime distributions may be heavy-tailed. volume 2583, pages 333–345, 2003.