Mach Learn (2006) 64:183–208
DOI 10.1007/s10994-006-7733-9

Randomised restarted search in ILP

Filip Železný · Ashwin Srinivasan · C. David Page Jr.

Received: 10 April 2005 / Revised: 14 November 2005 / Accepted: 11 February 2006 / Published online: 8 May 2006

Abstract Recent statistical performance studies 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 of the non-restarted randomised search exhibits a slower-than-exponential decay (that is, a "heavy tail"), restarts can reduce the search cost expectation. We report on an empirical study of randomised restarted search in ILP. Our experiments conducted on a high-performance distributed computing platform provide an extensive statistical performance sample of five search algorithms operating on two principally different classes of ILP problems, one represented by an artificially generated graph problem and the other by three traditional classification benchmarks (mutagenicity, carcinogenicity, finite element mesh design). 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), (2) estimate the conditional expected clause score given the cutoff value and the invested search cost, and (3) compare the performance of randomised restarted search strategies to a deterministic non-restarted search. Our findings indicate striking similarities across the five search algorithms and the four domains, in terms of the basic trends of both the statistics (1) and (2). Also, we observe that the cutoff value is critical for the performance of the search algorithm, and using its optimal value in a randomised restarted search may decrease the mean search cost (by several orders of

Editors: Rui Camacho

F. Železný (✉)
Czech Technical University, Prague, Czech Republic
e-mail: zelezny@fel.cvut.cz

A. Srinivasan
IBM India Research Laboratory, New Delhi, India
e-mail: ashwin.srinivasan@in.ibm.com

C. D. Page Jr.
University of Wisconsin, Madison, USA
e-mail: page@biostat.wisc.edu

magnitude) or increase the mean achieved score significantly with respect to that obtained with a deterministic non-restarted search.

Keywords Inductive logic programming · Randomized search · Monte carlo study

1. Introduction

Computer programs now collectively termed "Inductive Logic Programming" (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 Džeroski, 2001 for an overview) successful implementations have been relatively uniform, namely, engines that repeatedly examine sets of candidate rules to find the "best" ones.

Here we view ILP a discrete optimization problem where one maximizes, by means of search, an objective (scoring) function measuring the agreement of a rule with the sample data. The choice of search method can critically affect the performance of an ILP system on non-trivial problems. Enumerative search methods (such as the optimal branch-and-bound algorithm), despite their attractive simplicity, are not *robust* in the sense of achieving a balance between efficiency and efficacy across different problems (Goldberg, 1989). For many practical problems that engender very large spaces of discrete elements, enumerative search, however clever, becomes intractable. Up to special cases, where the organization of the search space together with the character of the objective function allow for efficient deterministic optimization, we are forced to take seriously Trefethen's Maxim No. 30 (Trefethen, 1998): "If the state space is huge, the only reasonable way to explore it is at random."

Research into time-critical reasoning (Zilberstein, 1998) led to the formalization of "anytime" algorithms that abandon optimality in favour of "good" solutions achieved using bounded resources. This goal was also followed by recent developments of efficient automatic model-checkers based on novel randomised search methods. Prominent examples are the GSAT and WalkSat methods checking the satisfiability of propositional formulae (Selman, Levesque & Mitchell, 1992), 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 (Gomes & Selman, 1999). 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 cost distribution 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 (Železný, Srinivasan & Page, 2003). There, a simple adaptation of a method known as Randomised Rapid Restarts (Gomes et al., 2000) 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 four domains, falling into two, principally different classes. The first class is represented by an artificially generated, noise-free, graph classification problem, in which the target theory can be modelled by a single, long clause (10 literals in the body). The second class consists of three traditional ILP benchmarks: Mutagenesis, Carcinogenesis and Finite Element Mesh Design. In these problems, good theories typically consist of multiple, relatively short clauses (typically up to 5 body literals). Although the natures of the two classes are quite different to each other, the main statistical findings relate equally to all the four domains.

Previous work on stochastic search in ILP appeared both in early ILP research (Kovačič et al., 1992) as well as in the recent paper (Serrurier, Prade & Richard, 2004). Unlike the present study, the mentioned approaches were concerned with adaptations of the simulated annealing optimization heuristic. Promising results were reported in learning problems including the domain of Finite Element Mesh Design (Pompe, Kovačič & Kononenko, 1993) as well as a musical domain (Pompe, Kononenko & Makše, 1996) (learning to compose two-voice counterpoint).

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 (Muggleton, 1995) 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 boldface (summarised in Table 1). All strategies include restarts (if γ is a finite value). Restarting DTD 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 being

$search(B, H, E, s^{suf}, c^{all}, \gamma)$: 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}, $(0 < c^{all} \leq \infty)$; and the maximum number of clauses evaluated on any single restart 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. $Active := \mathbf{UpdateActiveList}(Active, Ref)$
11. $Prune := \mathbf{Prune}(Active, S^*)$
12. $Active := Active \setminus Prune$
13. **Select** D^{curr} from $Active$; $Active := Active \setminus D^{curr}$
14. $Ref := \mathbf{Refine}_{B,H,(\gamma-N)}(D^{curr})$
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

chosen as the saturant.[1] A randomised implementation ('*R*') results in 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 Appendix A for more details on how this is achieved).

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).

[1] To avoid artifacts arising from any particular example ordering, in Section 3 we will measure the *average* performance of the DTD search variant, taking successively each positive example as the saturant.

Table 1 Implementation differences among 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 bi-directional refinement (specialisation and generalisation). See text for more details on these entries

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

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.

By means of choosing between a deterministic and a stochastic implementation of the respective **Select** directives in line 4 (starting clause selection) and 13 (next clause selection), the ILP algorithm simulates different search strategies suggested in previous general studies on restarted search. The remaining **Select** command, in line 3, determining what bottom clause is used to constrain the search space, is however special to ILP. We decided for a randomized version of the selection in all the restarted strategies in order to increase the variance between individual tries (a try is the sequence of steps 4–14 in Fig. 1). The decision will be formally justified in the following section.

2.2. Evaluation

Informally, given some clause evaluation function, for each search strategy we ask two questions. Firstly,

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

Here, we treat the number of search space nodes (clauses) explored as representative of the 'search cost' and quantify this cost-score trade-off by the expected value of the smallest cost needed to achieve or exceed a desired score s^{suf}.[2] Formally, we assume that $c^{all} := \infty$ and will investigate the statistical relationship between the random variables St, s^{suf}, γ, C which respectively denote the search strategy, the desired score, the number of clauses searched on a single restart, and the smallest cost needed to achieve or exceed the desired score. For each triple St, s^{suf}, γ, we wish to estimate the conditional cumulative distribution function $F(C|St, s^{suf}, \gamma)$ from which we will calculate the expectation $\mathbf{E}[C|St, s^{suf}, \gamma]$. For simplicity, we will denote this expectation as

$$\bar{C}(St, s^{suf}, \gamma) \tag{1}$$

As we will analyze each strategy individually, we will omit the St argument as a further notational simplification whenever its instantiation is clear from the context.

Given this evaluation measure, we can formally justify our choice of using a random bottom clause at each restart of a restarted strategy, by means of the following theorem, proved in Appendix C.

Theorem 1. *Let St_1 and St_2 be two instances of the algorithm* search *in Fig. 1, running on the same inputs and differing only in that the* **Select** *command on line 3 is*

- deterministic *for St_1, that is,* **Select** *yields the same saturant $e^+ \in E^+$ in all tries (e^+ has been drawn from E^+ with uniform probability prior to executing St_1).*
- stochastic *for St_2, that is, in each try* **Select** *draws e^+ randomly with uniform probability from E^+.*

Then $\bar{C}(St_1, s^{suf}, \gamma) \geq \bar{C}(St_2, s^{suf}, \gamma)$ for any s^{suf}, γ.

Our second informal question is

What clause score is achieved given an allocated search cost?

Here we deal with the random variables St, c^{all}, γ, S denoting respectively the search strategy, the allocated cost, the number of clauses searched on a single restart, and the highest score achieved during the search. Analogically to the previous question of interest, we assume that $s^{suf} := -\infty$ and will estimate the conditional expectation of S, denoted as

$$S(St, c^{all}, \gamma) \tag{2}$$

where we will again omit the St parameter whenever focusing on a single strategy.

Although there is an obvious Bayesian relationship[3] between C and S, there is no immediate relationship between their expectations. In Section 3, we will investigate this relationship by considering another statistic $C^*(s^{suf}, \gamma)$ defined as the smallest cost whose investment results in achieving the expected score s^{suf}, i.e. $C^*(s^{suf}, \gamma) = \min\{c^{all} | \bar{S}(c^{all}, \gamma) \geq s^{suf}\}$. Then we will give sufficient conditions under which $C^*(s^{suf}, \gamma) \leq \bar{C}(s^{suf}, \gamma)$.

The following points are evident, but worth restating:

[2] 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).

[3] Namely $P(C \leq c^{all} | S \geq s^{suf}) = P(S \geq s^{suf} | C \leq c^{all}) P(C \leq c^{all}) / P(S \geq s^{suf})$.

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).[4]
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. We will discuss this question further in Section 3.4, however, its full empirical assessment is 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 (Giordana & Saitta, 2000) 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.

3. Empirical evaluation

3.1. Materials

3.1.1. Data

Experiments were conducted using four ILP benchmarks. The first data set describes a set of 40 directed graphs. 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 (target) clause of 10 body literals. Besides the target clause, there exist multiple other clauses (subgraphs) in the search space cleanly separating the positive examples from the negatives. An example clause is shown below:

```
positive(A) :-  a(A,B,B), a(A,C,B), a(A,C,D),
                a(A,E,F), a(A,D,G), a(A,H,I),
                b(A,C), b(A,H), r(A,J), r(A,H).
```

Background knowledge facts `a(`g`,`$v1$`,`$v2$`)` state the existence of an edge from vertex $v1$ to vertex $v2$ in graph g, and `b(`g`,`v`)` (`r(`g`,`v`)`) denote that vertex v in graph g is coloured black (red).

The other three problems—Mutagenesis, Carcinogenesis and Finite Element Mesh Design—have been discussed extensively in ILP literature. As for Mutagenesis, we use here one of the datasets described in our previous publication, namely the data pertain-

[4] 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. In fact, even in the case of cover search we do not see a reason why the ranking should be different for multiple-rule learning than that estimated for single-rule learning, in other words, why the removal of a subset of positives should have a (un)favorable effect on any particular search method.

Table 2 Differences between the experimental data sets

Property	Graphs	Mutagenesis
Origin	Artificially generated	Biochemical literature
Noise	No	Yes
'Target' theory	Yes	No
'Good' theory	One long clause (10 lits)	Multiple clauses
# pos/neg examples	20/20	125/63
Property	**Carcinogenesis**	**Fnt Elm Mesh Design**
Origin	Biochemical literature	Engineering Designs
Noise	Yes	No
'Target' theory	No	No
'Good' theory	Multiple clauses	Multiple clauses
# pos/neg examples	182/148	223/242

ing to 188 "regression-friendly" compounds (Srinivasan et al.,1996). Table II describes the principal differences between the data sets. The graph and mesh data sets are available on request to the first author. The software for the graph data generation can be obtained from the third author. The mutagenesis and carcinogenesis dataset can be obtained via anonymous ftp to `ftp.comlab.ox.ac.uk` in the directories `pub/Packages/ILP/Datasets/mutagenesis/aleph`.

3.1.2. Algorithms 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 conducted on the Condor computer cluster at the University of Wisconsin in Madison (Mutagenesis and Graphs) and on an SGI Altix 3700 supercomputer at the Czech Technical University in Prague (Carcinogenesis and Mesh). 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 values $\bar{C}(s^{suf}, \gamma)$ and $\bar{S}(c^{all}, \gamma)$ for each of the five strategies in Section 2.1. A straightforward way to collect the required statistical sample needed to estimate the respective 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 (cost) parameter s^{suf} (c^{all}) and the restart cutoff parameter γ, each time recording the resulting value of C (S). This approach would however perform a lot of redundant computation. Instead we adopt the following method:

For each problem (Graphs, Mutagenesis, Carcinogenesis, Mesh)

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. $B \cup \{D\} \models E_n$. 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 expectations $\bar{C}(s^{suf}, \gamma)$ for any s^{suf}, γ and $\bar{S}(c^{all}, \gamma)$ for any c^{all}, γ. In Appendix B we describe how the sample of 6,000 performance vectors can be used to obtain an unbiased estimate of these expectations.

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 B shows how these performance vectors can be used to obtain a biased (lower bound) estimate of $\bar{C}(s^{suf}, \gamma = \infty)$ for any s^{suf} and an unbiased estimate of $\bar{S}(c^{all}, \gamma = \infty)$ for any c^{all}.

3.3. Results

Figures 2–5 show diagrammatically for the randomized strategies (1) the estimated expected number of evaluated clauses ('expected cost value') as a function of the pre-set sufficient score and the restart cutoff parameter, and (2) the estimated expected score achieved as a function of the pre-set number of evaluated clauses and the restart cutoff. The total number of such generated diagrams is 16 (four domains, four randomized strategies), out of which we are showing here the four 'diagonal' combinations (Graphs-RTD, Mutagenesis-GSAT, Carcinogenesis-WSAT, Mesh-RRR). As for a given domain, cost and score expectations of the four restarted strategies turned out roughly similar, this sample is sufficient to illustrate the principal trends observed.

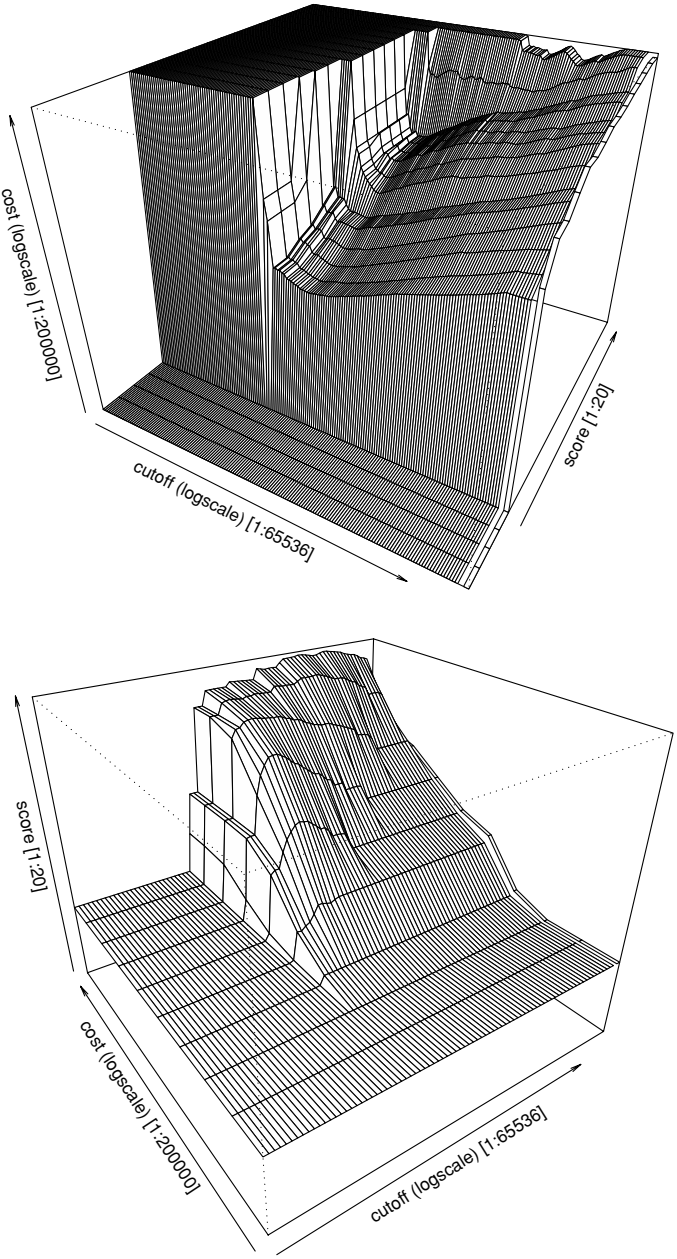

Fig. 2 RTD on Graphs: expected costs (upper diagram) and expected scores (lower diagram)

Besides the mesh corresponding to the given restarted randomized strategy, each plot also shows the expected value for the non-restarted (thereby independent of the cut-off parameter γ) DTD strategy, by an 'isolated' strip located at the maximal cutoff coordinate.

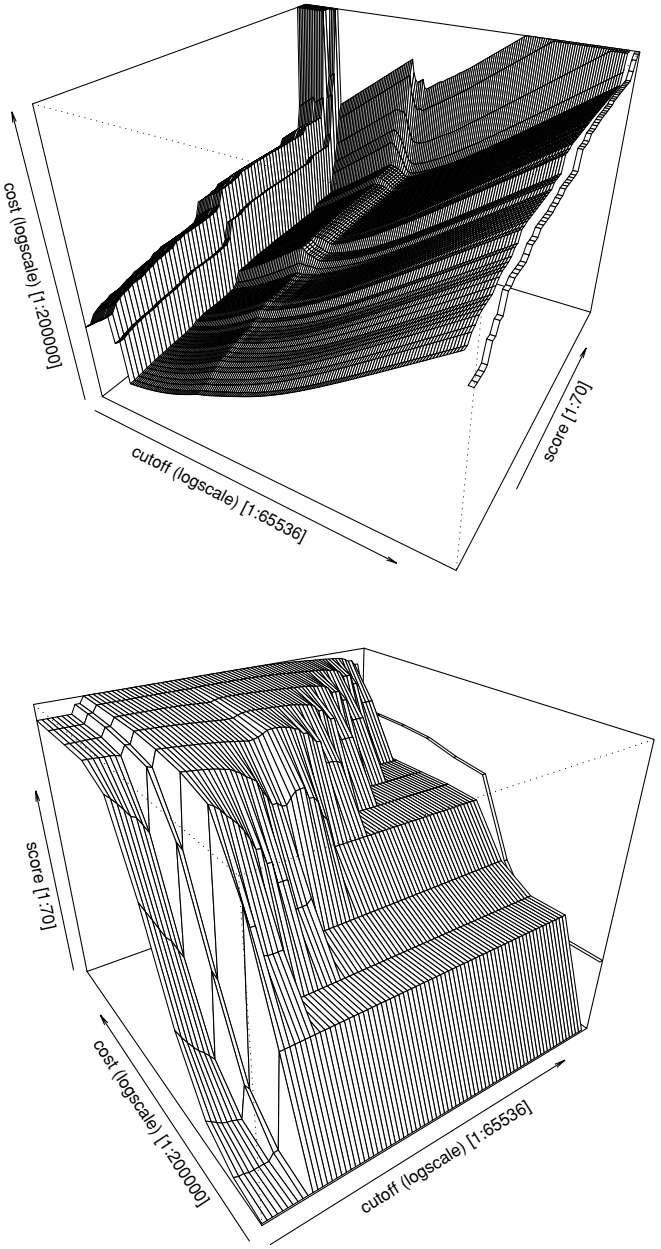

Fig. 3 GSAT on Mutagenesis: expected costs (upper diagram) and expected scores (lower diagram)

In all the upper (expected-cost) diagrams, the highest plateau 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 (z) coordinate c_{max}.

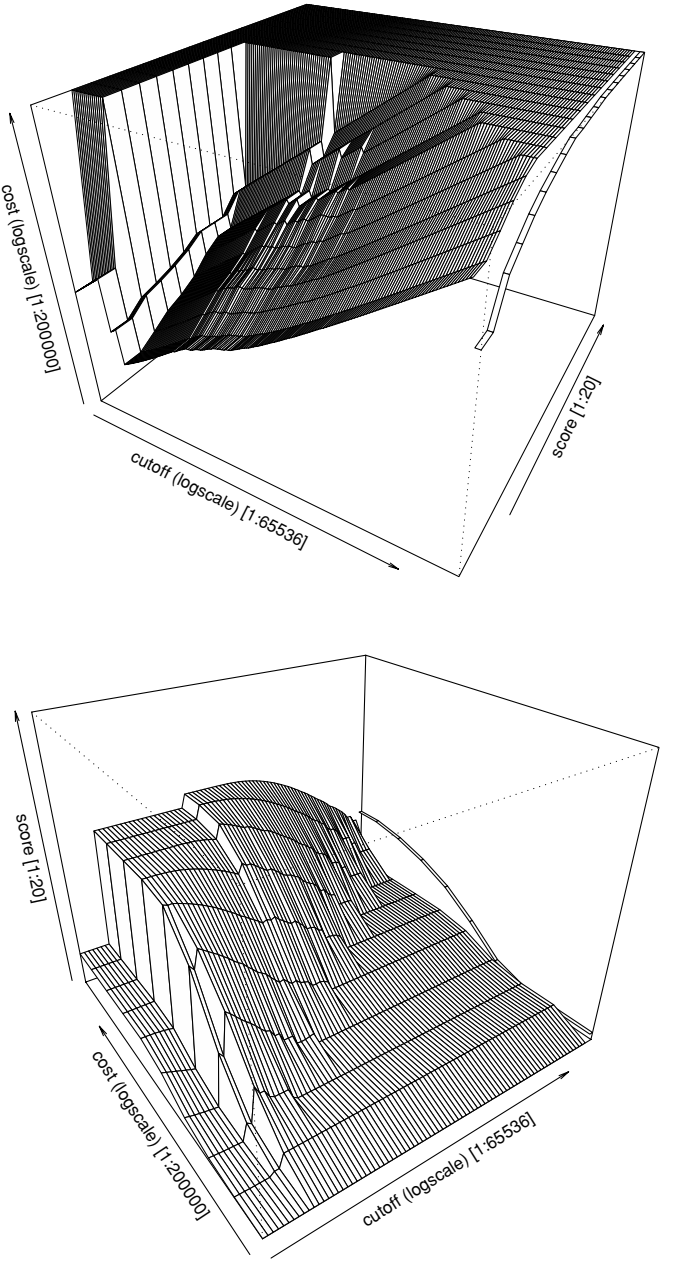

Fig. 4 WSAT on Carcinogenesis: expected costs (upper diagram) and expected scores (lower diagram)

3.3.1. Basic trends

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:

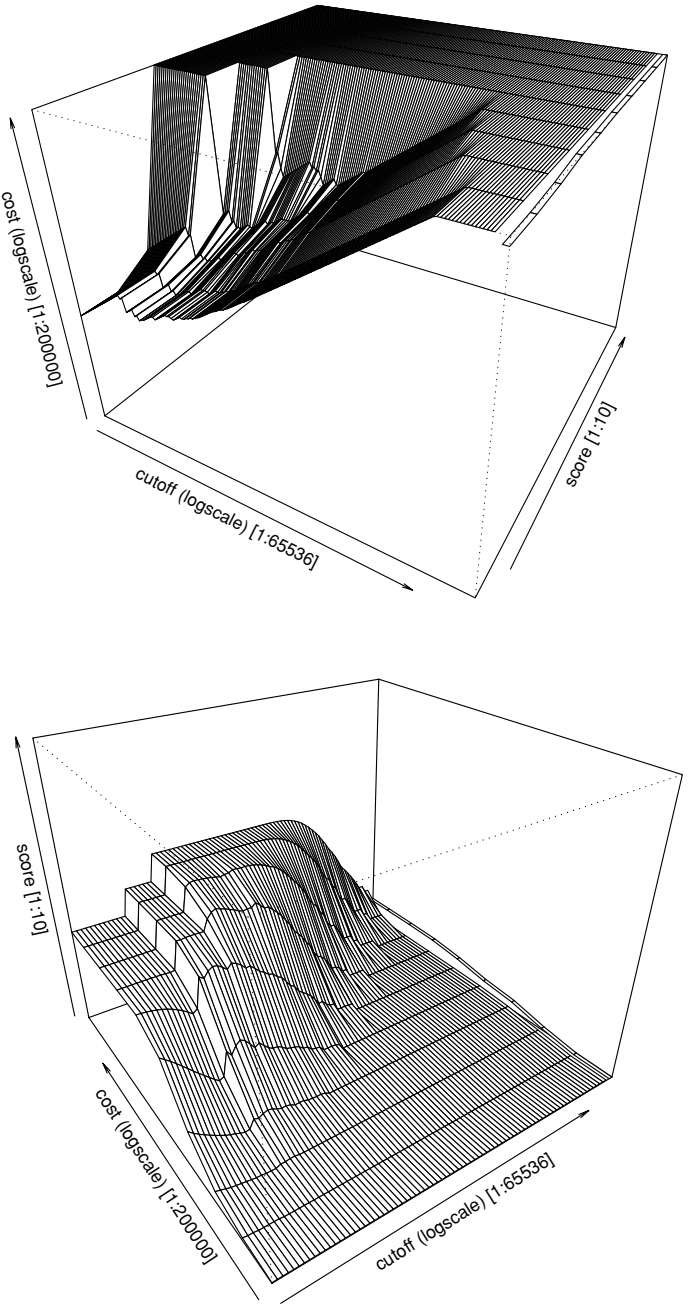

Fig. 5 RRR on Mesh: expected costs (upper diagram) and expected scores (lower diagram)

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 1,000 times for mutagenesis. A significant impact is also observed on the expected score.

2. $\gamma \approx 100$ is a 'good' choice for all the investigated strategies in all the domains. For RTD, the value is close to the optimum for most of the s^{suf} settings. The other restarted strategies, which unlike RTD begin the search in the interior of the search lattice, the optimum γ for most s^{suf} is lower (slightly above 10 clauses explored), but also here a local minimum of the expected cost at $\gamma \approx 100$ frequently emerges.
3. $\gamma < 10$ appears to be uniformly a 'bad' choice for all randomised strategies.
4. For both domains, other than very high values of s^{suf}, the expected costs of the restarted strategies are uniformly lower than that of the non-restarted strategy DTD for a wide range of γ ($100 \leq \gamma \leq 10,000$).

3.3.2. *Relationships between the expected values*

We now investigate the relationship between $\bar{C}(s^{suf}, \gamma)$ and $\bar{S}(c^{all}, \gamma)$. Consider the quantity

$$C^*(s^{suf}, \gamma) = \min\{c^{all} | \bar{S}(c^{all}, \gamma) \geq s^{suf}\}. \tag{3}$$

While $\bar{C}(s^{suf}, \gamma)$ captures the expected number of explored clauses needed to achieve score s^{suf}, $C^*(s^{suf}, \gamma)$ corresponds to the smallest number of clauses whose exploration results in the expected achieved score of at least s^{suf}. We plot the empirically measured values of both these quantities for the four considered method-domain combinations in Fig. 7. For clarity of comparison, we show the two plots for a single cutoff $\gamma = 1000$, a value yielding good performance of the restarted methods in terms of the cost or score expectation functions while avoiding their extremal points (such as at $\gamma \approx 100$). For all four method-domain combinations and a vast majority of γ and s^{suf} values (although not all of them), we observe a systematic

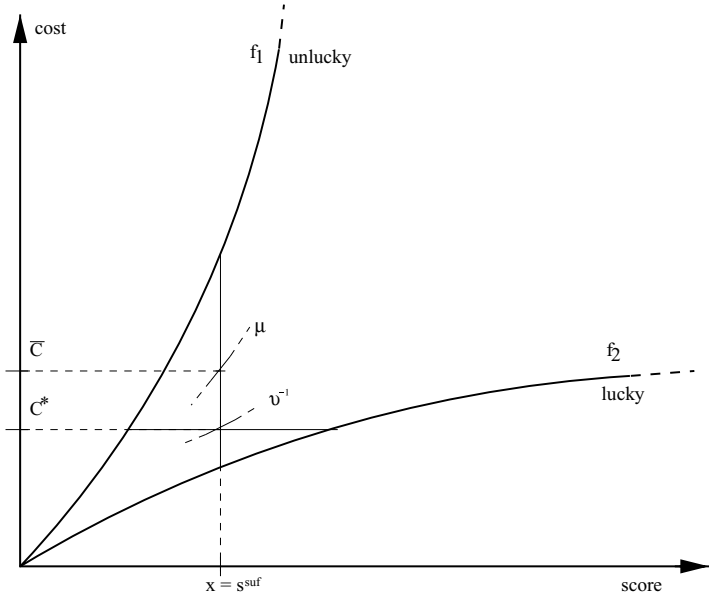

Fig. 6 Representing performance vectors through continuous functions; see text for details

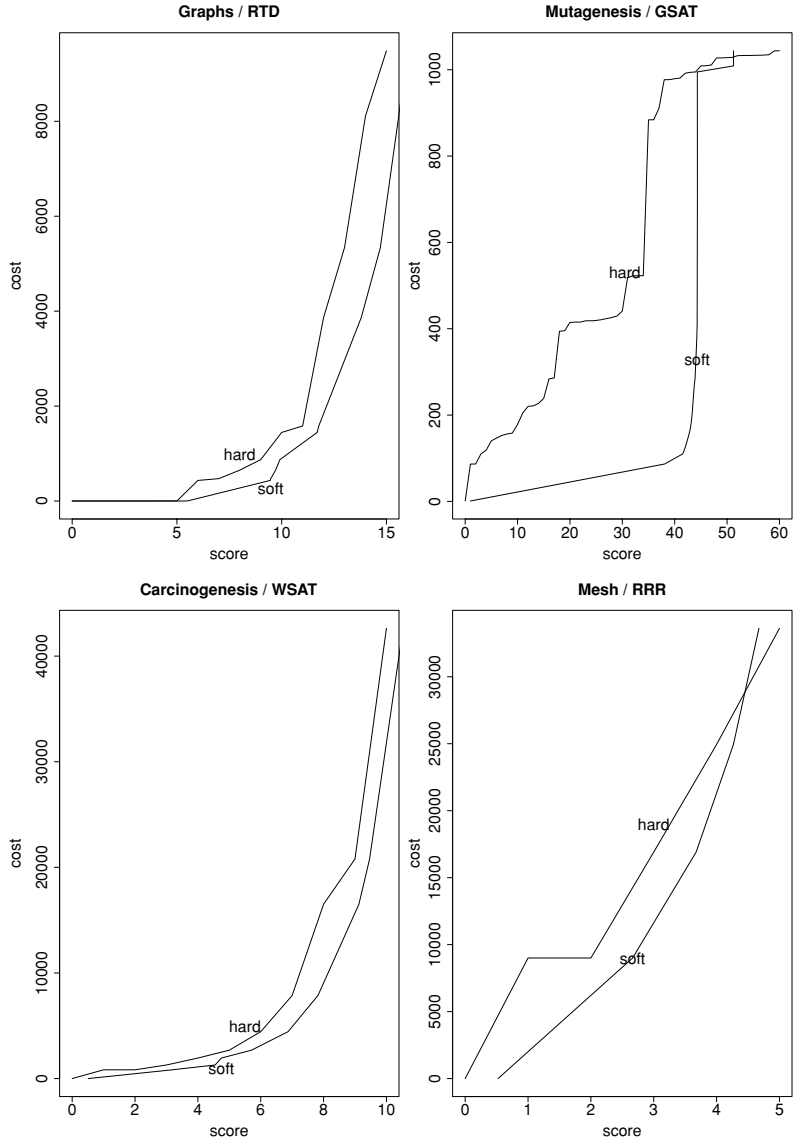

Fig. 7 Values of $\bar{C}(s^{suf}, \gamma)$ (denoted as "hard"), and $C^*(s^{suf}, \gamma)$ (denoted as "soft"), where $\gamma = 1000$. The *score* axes are trimmed to values corresponding to finite costs

relationship

$$\bar{C}(s^{suf}, \gamma) > C^*(s^{suf}, \gamma) \tag{4}$$

We offer a tentative theoretical model providing at least a partial insight into this empirical phenomenon. As the analysis does not depend on γ, we will omit this parameter here. For simplicity of proving, we now view performance vectors as bijective continuous 'performance functions' $f_i : \mathcal{R}^+ \cup \{0\} \to \mathcal{R}^+ \cup \{0\}$. Figure 6 illustrates the situation for a case where the search has been executed twice, yielding two performance vectors, here represented by the

continuous functions f_1 and f_2. The cost $\bar{C}(s^{suf})$ then corresponds to the average of f_1 and f_2 at point $x = s^{suf}$ (see the vertical line stemming from x on the score axis). The cost $C^*(s^{suf})$ (smaller than $\bar{C}(s^{suf})$) corresponds to the vertical coordinate where $x = s^{suf}$ is the average of the functions inverse to the functions f_1 and f_2 (see the horizontal line stemming from C^* on the cost axis). For such calculated $\bar{C}(s^{suf})$ and $C^*(s^{suf})$, both being functions of s^{suf}, we will also introduce their respective continuous counterpart functions μ and ν (see again Fig. 6). In the general case of more than two performance functions, μ represents their average on the vertical axis, while ν represents the average of their inverse functions on the horizontal axis.

Our model further assumes the fundamental property of search problems characterized by heavy-tailed search cost distributions: there is a significant amount of both 'lucky' runs and 'unlucky' runs (Chen, Gomes & Selman 2001). In our model we assume the set of experimental executions of the randomized search (each corresponding to one performance function) can be partitioned into two disjoint sets: lucky and unlucky runs. The unlucky (lucky, respectively) runs have a convex (non-convex) shape of the performance function (remember that this function captures the growing number of explored clauses with growing score in a particular run). The example shown in Fig. 6 captures one lucky and one unlucky run. Our model does not directly impose specific assumptions on the quantities of the lucky or unlucky runs, but it does stipulate a form of balance by assuming that the inverse of the performance function $f_i^{-1}(y)$ of an unlucky (lucky) run i (i.e. its score achieved with cost y) is at all points y smaller (greater or equal) than the mean $\nu(y)$ among all runs. The following theorem (proved in Appendix C) says that Ineq. 4 is implied by these conditions.

Theorem 2. *Let $D = R^+ \cup \{0\}$ and $\{f_1, \ldots f_m, f_{m+1}, \ldots, f_n, \mu, \nu\}$ be a finite set of bijective functions $D \to D$ such that $\mu(x) \equiv \frac{1}{n}\sum_{i=1}^{n} f_i(x)$, $\nu(x) \equiv \frac{1}{n}\sum_{i=1}^{n} f_i^{-1}(x)$ and for all $1 \leq i \leq n$, $x \in D$: $f_i(0) = 0$, $f_i''(x)$ exists, and the following holds iff $i \leq m$: $f_i^{-1}(x) < \nu(x)$, $f_i''(x)$ is positive. Then $\mu(x) > \nu^{-1}(x)$ for all $x \in D$.*

We leave it for future work to determine how accurately the model assumptions capture the properties of the measured performance vectors and which of the assumptions are not met at the points where the inequality 4 does not hold in the empirical data.

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).

Our unit of cost was the number of clauses searched rather than cpu time. The advantages of such a choice are evident: besides making the results independent of any particular hardware platform used, the expected number of clauses tested in the search process is of interest in its own right. For example, it is linked to generalization performance as shown in Domingos (1999a) and discussed later in this section.

However, two issues require further investigation before conclusions can be made concerning the ranking of the strategies in terms of cpu time. First, DTD and RTD always begin 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 usually means spending less total evaluation time for the same number of clauses scored. To at least roughly assess how significantly this fact would change our results if cpu time was viewed as the cost, we performed an additional experiment on the Mutagenesis domain

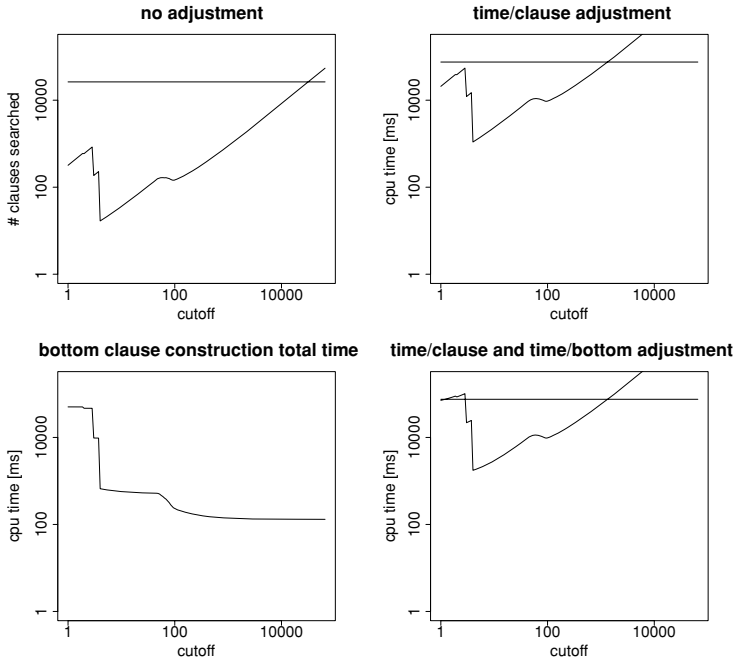

Fig. 8 GSAT on Mutagenesis, $s^{suf} = 35$: expected number of clauses searched (upper-left), expected time taken ignoring time for bottom clause construction (upper-right), expected total time spent on bottom clause construction (lower-left) and expected total time taken (lower-right). The horizontal line corresponds in all panes to DTD

estimating the average cpu time taken on evaluating a single clause for the respective search strategies. The measurement procedure followed the regime described in 3.2, except that besides the performance vector c_i recording the cost (growing number of clauses searched) we also recorded a corresponding vector of current processor times t_i. For each execution i we then calculated

$$t_i = \frac{t_i^l - t_i^1}{c_i^l} \tag{5}$$

where c_i^l (t_i^l) is the last element of c_i (t_i) and t_i^1 is the cpu time at the beginning of the execution of run i ($c_i^1 = 0$). Averaging over of t_i's, we obtained the estimate of cpu time per one clause evaluation, amounting to[5] 2.25 ms (standard deviation 1.15) for DTD, 2.86 ms (st. dev. 1.26) for RTD and 65.25 ms (st. dev. 59.83) for GSAT (WSAT and RRR differing only insignificantly from GSAT). Figure 8 (upper-left pane) shows a projection of the Mutagenesis cost-score Fig. 3 for the score $s^{suf} = 35$ (half of the maximum score), comparing the GSAT strategy with DTD (horizontal line). The upper-right pane in the Figure shows instead the cpu time obtained by multiplying the number of clauses searched by the time-per-clause estimate obtained for GSAT and DTD, respectively.

[5] All cpu times further reported were measured on a 64-bit single-processor computer with 3 GHz of system clock frequency.

Second, our measurement scheme did not assign any cost to the computation needed to construct a bottom clause for each restart. To take this factor into account, we calculated the average time per bottom clause construction, which in Mutagenesis amounted to 158 ms (st. dev. 38.04). Then for a given cutoff γ, each restarted method incurs an additional expected cpu time overhead

$$\frac{\bar{C}(s^{suf}, \gamma)}{\gamma} \cdot 158[ms] \tag{6}$$

(where the left fraction expresses the expected total number of restarts made). The lower-left pane of Fig. 8 shows the expected total time spent of bottom clause construction by GSAT and, finally, the lower-right pane shows the expected total cpu time where both time per clause evaluation and time per bottom construction are accounted for.

Although the large superiority of GSAT (as well as RRR and WSAT characterized by similar time overheads) w.r.t. DTD clearly reduces when changing the cost unit from the amount of evaluated clauses to cpu time spent, a significant gap in favor of the restarted methods is maintained in the neigbourhoods of both the cost minimum around $\gamma = 100$ (a local minimum we have observed in all tested domains) as well the one close to $\gamma = 10$ (a minimum characteristic for Mutagenesis).

Although this study was not concerned with direct measurement of the generalization performance of the learning algorithms, two evident factors may create variance in this respect among the five search methods: (i) the complexity of rules produced, and (ii) the volume of search space explored by the respective methods. Penalizing model complexity is a traditional technique used to avoid overfitting (Hastie, Tibshirani & Friedman 2001), which would suggest that the general-to-specific (DTD and RTD) methods biased towards simpler clauses are less prone to overfitting than the rest of the strategies (RRR, GSAT, WSAT). There is however mounting evidence (Domingos, 1999a) that the risk of overfitting grows in fact with the latter factor, i.e. the total number of models tested in the search process, with the representational complexity being only indirectly linked to overfitting (Domingos, 1999b) (as learning algorithms usually avoid complex models when reducing the search space volume). As a result, we have reason to expect those strategies examining on average fewer clauses to achieve a given training-data fit (i.e. all the restarted strategies with a suitable cutoff value) to generalize better than those examining more clauses (DTD).

It is encouraging that four apparently very different domains and all restarted strategies have yielded a similar range of 'good' values for the γ parameter. However, the plots, especially for 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 (Kautz et al., 2002) and may result in a more robust search algorithm.

4. Concluding remarks

Search is at the heart of most modern Inductive Logic Programming systems, and most have thus far employed well-known deterministic search methods.[6] In other fields confronted

[6] We note the exceptions of kernel-based approaches (Gaertner et al., 2004), which are not based on search as understood in this paper, and (Kovačič et al., 1992; Serrurier et al., 2004), which employ stochastic, rather than deterministic search.

with very large search spaces, there is now substantial evidence that the use of randomised restarted strategies yield superior results to deterministic ones (often making the difference between getting a good solution, or none at all). Randomised search strategies thus seem to represent a suitable approach to achieve 'any-time' ILP algorithms (Zilberstein, 1998), able to efficiently trade off between the quality of results and the resources consumed.

In 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 be a 'good' value for γ that works adequately 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; and on problems where the target hypotheses lie in 'difficult' locations identified by the research in Botta et al. (2003).

Appendix

A. Randomized start clause selection

Here we address the method for the random selection of a start clause at each restart, employed by all the considered search strategies except DTD and RTD.

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 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 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 jth 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

$estimate(\perp, \mathcal{L}, l, s)$: Given a most specific clause $\perp$ 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 $\perp$. The estimate is obtained from a sample of size s.

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

Fig. 9 A procedure for estimating the number of "legal" clauses of length $l > 1$. The estimate obtained in Step 3 above is unbiased. 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 $s = z^2_{\alpha/2}/(4e^2)$. Here z represents the standard normal variable as usual

efficient by various devices. Some examples are: (a) take C to be 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 $\perp$ 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. 9 for estimating them.

B. Calculating expected values from performance vectors

We describe here a technique for estimating the conditional expectations $\bar{C}(s^{suf}, \gamma)$ and $\bar{S}(c^{all}, \gamma)$, defined in Section 2.2, for each of the five strategies in Section 2.1. We exploit the fact that the expectations for arbitary s^{suf}(c^{all}, respectively) 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 of $\bar{C}(s^{suf}, \gamma)$ for DTD, but will still allow us to obtain unbiased estimates for restarted strategies for all values of $\gamma < c^{all}$.

B.1. Calculating the expected cost

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, $\bar{C}(s^{suf}, \gamma = \infty)$ 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 DTD will represent a *lower bounds* of its expected cost.

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

$$F(\gamma|s^{suf})\left(1 - F_s(\gamma|s^{suf})\right)^{t-1} \tag{7}$$

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 or fewer clauses. It is estimated for a given score s from the empirical data as the fraction

$$F(x|s) \approx \frac{|\{c_i|c_i(s) \leq x\}|}{|\{c_i\}|} \tag{8}$$

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 estimate remains unbiased.

Denoting the expected number of tries initiated before achieving s^{suf} as $\bar{T}(s^{suf}, \gamma)$,

$$\bar{T}(s^{suf}, \gamma) = F(\gamma|s^{suf}) \sum_{t=1}^{\infty} t \left(1 - F(\gamma|s^{suf})\right)^{t-1} \tag{9}$$

It equals 1 for $F(\gamma|s^{suf}) = 1$ and for $F(\gamma|s^{suf}) = 0$ we set $\bar{T}(s^{suf}, \gamma) = \infty$. If $0 < F(\gamma|s^{suf}) < 1$, it can be shown that Expression 9 converges to

$$\bar{T}(s^{suf}, \gamma) = \frac{1}{F(\gamma|s^{suf})} \tag{10}$$

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

$$\bar{C}(s^{suf}, \gamma) = \gamma \bar{T}(s^{suf}, \gamma) = \frac{\gamma}{F(\gamma|s^{suf})} \tag{11}$$

However, $\bar{C}(s^{suf}, \gamma)$ 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

$$\gamma \cdot (\bar{T}(s^{suf}, \gamma) - 1) = \gamma \cdot \left(\frac{1}{F(\gamma|s^{suf})} - 1\right) \tag{12}$$

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

$$D_t(n) = P(N = n|(t-1)\gamma < C \leq t\gamma, s^{suf}, \gamma) \tag{13}$$

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, i.e. 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[7] that $N = C$, we can write

$$D_1(n) = P(C = n | C \leq \gamma, s^{suf}) \tag{14}$$

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

$$D_1(n) = \frac{P(C = n, C \leq \gamma | s^{suf})}{P(C \leq \gamma | s^{suf})} \tag{15}$$

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

$$D_1(n) = \frac{P(C = n | s^{suf})}{P(C \leq \gamma | s^{suf})} = \frac{F(n|s^{suf}) - F(n-1|s^{suf})}{F(\gamma|s^{suf})} \tag{16}$$

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

$$\sum_{n=1}^{\infty} n D_t(n) = \sum_{n=1}^{\gamma} n D_1(n) = \sum_{n=1}^{\gamma} n \frac{F(n|s^{suf}) - F(n-1|s^{suf})}{F(\gamma|s^{suf})} \tag{17}$$

Denoting this value by $L_F(\gamma|s^{suf})$ and adding it to Eq. (12), we get the expected total number of evaluated clauses:

$$\bar{C}(s^{suf}, \gamma) = \gamma \cdot \left(\frac{1}{F(\gamma|s^{suf})} - 1 \right) + L_F(\gamma|s^{suf}) \tag{18}$$

Recall that the conditional distribution $F(.|.)$ used above can be estimated from the performance vectors as described by Eq. 8.

B.2. Calculating the expected score

Here we are concerned with estimating the expectation $\bar{S}(c^{all}, \gamma)$. The probability $P(S \geq s | c^{all}, \gamma)$ of achieving at least score s by exploring c^{all} clauses (always restarting after γ clauses explored) is equal to the probability $P(C \leq c^{all} | s, \gamma)$ of exploring at most c^{all} clauses until one with at least the score s is found.

For the non-restarted DTD, $P(C \leq c^{all} | s, \gamma) = F(c^{all}|s)$, estimated without bias from $|E^+|$ performance vectors (each corresponding to one possible saturant) using Eq. 8. The

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

expected score can then be calculated as

$$\bar{S}(c^{all}) = \sum_{s=1}^{s_{max}} s \left[F(c^{all}|s) - F(c^{all}|s+1) \right] \tag{19}$$

For a restarted strategy with cutoff γ, consider that the algorithm conducts $c^{all} \div \gamma$ (integer division) complete tries and then spends $c^{all}\ mod\ \gamma$ (integer division remainder) of cost in a non-restarted search. Then

$$P(S \geq s|c^{all}, \gamma) = P(C \leq c^{all}|s, \gamma) = 1 - P(C > c^{all}|s, \gamma) \tag{20}$$

where the probability $P(C > c^{all}|s, \gamma)$ corresponds to the simultaneous occurrence of $c^{all} \div \gamma$ independent events of not achieving score s within a single complete try (each such event has the probability $1 - P(C \leq \gamma|s) = 1 - F(\gamma|s)$) and one event of not achieving score s exploring $c^{all}\ mod\ \gamma$ clauses in a non-restarted search (this event has the probability $1 - P(C \leq c^{all}\ mod\ \gamma|s) = 1 - F(c^{all}\ mod\ \gamma|s)$). That is

$$P(S \geq s|c^{all}, \gamma) = 1 - [1 - F(\gamma|s)]^{c^{all} \div \gamma} \left[1 - F(c^{all}\ mod\ \gamma|s) \right] \tag{21}$$

which we abbreviate by $\epsilon(s, c^{all}, \gamma)$. To obtain the expected score, one again estimates the values $F(.|.)$ by Eq. (8) and calculates

$$\bar{S}(c^{all}, \gamma) = \sum_{s=1}^{s_{max}} s \left[\epsilon(s, c^{all}, \gamma) - \epsilon(s-1, c^{all}, \gamma) \right] \tag{22}$$

C. Proofs of theorems

Lemma 1 (Arithmetic-harmonic means inequality). *Let* $a_1, a_2, \ldots a_n \in \mathcal{R}^+$, $n \geq 1$. *Then*

$$\frac{1}{n} \sum_{i=1}^{n} a_i \geq \left(\frac{1}{n} \sum_{j=1}^{n} \frac{1}{a_j} \right)^{-1} \tag{23}$$

where the means are equal if and only if all a_i *are equal.*

Proof: Can be found in many mathematical handbooks, see eg. (Kenney & Keeping, 1962). It follows from the well known arithmetic-geometric-harmonic means inequality. □

Theorem 1. *Let* St_1 *and* St_2 *be two instances of the algorithm* search *in Fig. 1, running on the same inputs and differing only in that the* **Select** *command on line 3 is*

- deterministic *for* St_1*, that is,* **Select** *yields the same saturant* $e^+ \in E^+$ *in all tries (*e^+ *has been drawn from* E^+ *with uniform probability prior to executing* St_1*).*
- stochastic *for* St_2*, that is, in each try* **Select** *draws* e^+ *randomly with uniform probability from* E^+*.*

Then $\bar{C}(St_1, s^{suf}, \gamma) \geq \bar{C}(St_2, s^{suf}, \gamma)$ *for any* s^{suf}, γ.

Proof: For St_1, we can condition the expected cost expressed in Eq. (18) on the positive saturated example e_i^+ drawn from $E^+ = \{e_1^+, e_2^+ \dots e_p^+\}$, $p \geq 1$, randomly with uniform probability $P(e_i^+) = 1/p$ prior to execution of St_1:

$$\bar{C}(St_1, s^{suf}, \gamma) = \frac{1}{p} \sum_{(index)=1}^{p} \bar{C}(s^{suf}, \gamma, e_i^+] \tag{24}$$

$$= \gamma \left(\frac{1}{p} \sum_{(index)=1}^{p} \frac{1}{F(\gamma | s^{suf}, e_i^+)} - 1 \right) \tag{25}$$

$$+ \frac{1}{p} \sum_{(index)=1}^{p} L_F(\gamma | s^{suf}, e_i^+) \tag{26}$$

where $L_F(\gamma | s^{suf}, e^+)$ abbreviates Expression 17 (with all conditional parts extended by the e^+ condition). For St_2, there is no saturant selected prior to execution, but here the expected cost from Eq. (18) can be expressed as

$$\bar{C}(St_2, s^{suf}, \gamma) = \gamma \left(\frac{p}{\sum_{(index)=1}^{p} F(\gamma | s^{suf}, e_j^+)} - 1 \right) \tag{27}$$

$$+ \frac{1}{p} \sum_{(index)=1}^{p} L_F(\gamma | s^{suf}, e_k^+) \tag{28}$$

In 27 we decomposed $F(\gamma | s^{suf})$ by conditioning on the example e_j^+ saturated at the beginning of a non-terminal try, while in 28 we conditioned $L_F(\gamma | s^{suf})$ on the example e_k^+ saturated in the last try.

Comparing 25 with 27 and 26 with 28, we get that $\bar{C}(St_1, s^{suf}, \gamma) \geq \bar{C}(St_2, s^{suf}, \gamma)$ whenever the inequality holds between the respective left terms in the parentheses in 25 and 27, namely

$$\frac{1}{p} \sum_{(index)=1}^{p} \frac{1}{F(\gamma | s^{suf}, e_i^+)} \geq \frac{p}{\sum_{(index)=1}^{p} F(\gamma | s^{suf}, e_j^+)} \tag{29}$$

Taking the inverse value of both sides of the inequality and reverting its sign, the inequality follows from Lemma 1. □

A further, obvious consequence of Lemma 1 is that the expected search costs of St_1 and St_2 are equal if and only if the probability $F(\gamma | s^{suf}, e^+)$ of achieving at least score s^{suf} exploring γ clauses using the bottom clause obtained from saturating e^+ is the same for all positive examples $e^+ \in E^+$. Otherwise St_1 will incur a larger expected cost that the "more randomized" strategy St_2.

Theorem 2. *Let* $D = R^+ \cup \{0\}$ *and* $\{f_1, \dots f_m, f_{m+1}, \dots, f_n, \mu, \nu\}$ $(1 < m < n)$ *be a finite set of bijective functions* $D \to D$ *such that* $\mu(x) \equiv \frac{1}{n} \sum_{i=1}^{n} f_i(x)$, $\nu(x) \equiv \frac{1}{n} \sum_{i=1}^{n} f_i^{-1}(x)$

and for all $1 \leq i \leq n$, $x \in D$: $f_i(0) = 0$, $f_i''(x)$ *exists, and the following holds iff* $i \leq m$: $f_i^{-1}(x) < \nu(x)$, $f_i''(x)$ *is positive. Then* $\mu(x) > \nu^{-1}(x)$ *for all* $x \in D$.

Proof: Since $\mu(0) = \nu^{-1}(0) = 0$, it suffices to show that $\mu'(x_0) \geq \nu^{-1\prime}(x_0)$ for all $x_0 \in D$. Here $\mu'(x_0) = \frac{1}{n}\sum_{i=1}^{n} f'(x_0)$. Applying twice the rule of inverse function differentiation, we also get

$$\nu^{-1\prime}(x_0) = \frac{1}{\nu'(y_0)} = \frac{n}{\sum_{i=1}^{n} f_i^{-1\prime}(y_0)} = \frac{n}{\sum_{i=1}^{n} \frac{1}{f_i'(x_i)}} \tag{30}$$

where $y_0 = \nu^{-1}(x_0)$ and $x_i = f_i^{-1}(y_0)$. We further decompose $\nu^{-1\prime}(x_0)$

$$\nu^{-1\prime}(x_0) = \frac{n}{\sum_{i=1}^{m} \frac{1}{f_i'(x_i)} + \sum_{j=m+1}^{n} \frac{1}{f_j'(x_j)}} \tag{31}$$

For $1 \leq i \leq m$, $y_0 \in D$, we assumed that $f_i^{-1}(y_0) < \nu(y_0)$, that is, $x_i < x_0$. Similarly $x_j > x_0$ for $m < j \leq n$. Due to the positivity of f_i'' and non-positivity of f_j'', we can bound $f_i'(x_i) < f_i'(x_0)$ and $f_j'(x_j) \leq f_j'(x_0)$, so

$$\nu^{-1\prime}(x_0) < \frac{n}{\sum_{i=1}^{m} \frac{1}{f_i'(x_0)} + \sum_{j=m+1}^{n} \frac{1}{f_j'(x_0)}} = \frac{n}{\sum_{k=1}^{n} \frac{1}{f_k'(x_0)}} \tag{32}$$

To show that indeed $\mu'(x_0) > \nu^{-1\prime}(x_0)$, it remains to verify the inner inequality in:

$$\mu'(x_0) = \frac{1}{n}\sum_{i=1}^{n} f_i'(x_0) \geq \frac{n}{\sum_{k=1}^{n} \frac{1}{f_k'(x_0)}} > \nu^{-1\prime}(x_0) \tag{33}$$

which, however, follows instantly from Lemma 1. □

Acknowledgments The authors would like to thank the ILP 2004 referees as well as the MLJ special issue referees for their informative suggestions. A.S. would like to acknowledge the generous support provided by the Computing Laboratory, Oxford University during the course of this work and for continuing to act as the primary source for the Aleph program. F. Ž is supported by the Grant Agency of the Czech Academy of Sciences through the project KJB201210501. Part of this study was conducted during F. Ž.'s stay at the Dept. of Biostatistics at UW Madison in 9-10/2005, enabled by the project MSM 1P05ME755 of the Czech Ministry of Education.

The Condor Software Program (Condor) was developed by the Condor Team at the Computer Sciences Department of the University of Wisconsin-Madison. All rights, title, and interest in Condor are owned by the Condor Team. Analysis of collected data was possible thanks to the open-source "R" System for Statistical Computing. Access to SGI Altix 3700 was kindly provided by the Supercomputing Services Center of the Czech Technical University in Prague.

References

Botta, M., Giordana, A., Saitta, L., & Sebag, M. (2003). Relational learning as search in a critical region. *Journal of Machine Learning Research*, (4), 431–463.

Chen, H., Gomes, C. P., & Selman, B. (2001). Formal models of heavy-tailed behavior in combinatorial search. *Proceedings of the 7th international conference on principles and practice of constraint programming.* Springer-Verlag, (pp. 408–421).

Domingos, P. (1999a). Process-oriented estimation of generalization Error. *IJCAI99* (pp. 714–721).
Domingos, P. (1999b). The role of Occam's Razor in knowledge discovery. *Data Mining and Knowledge Discovery*, *3*, 409–425.
Džeroski, S. (2001). Relational data mining applications: An Overview. *Relational data mining.* Springer-Verlag (pp. 339–364).
Gaertner, T., Lloyd, J. W., & Flach, P. A. (2004). Kernels and distances for structured data. *Machine Learning 57*(3), 205–232.
Giordana, A., & Saitta, L. (2000).Phase transitions in relational learning. *Machine Learning*, *41*(2), 217–251.
Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning*. Addison-Wesley.
Gomes, C., & Selman B. (1999). On the fine structure of large search spaces. *Proceedings the eleventh international conference on tools with artificial intelligence*.
Gomes, C., Selman, P. B., Crato, N., & Kautz, H. A. (2000). Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning 24*(1/2), 67–100.
Hastie, T., Tibshirani, R., & Friedman, J. (2001). *The elements of statistical learning: Data mining, inference, and prediction.* Springer.
Kautz, H., Horvitz, E., Ruan, Y., Gomes, C., & Selman, B. (2002). Dynamic restart policies. *Proceedings of the eighteenth national conference on artificial intelligence*.
Kenney, J. F., & Keeping, E. S. (1962). *Harmonic mean*. Van Nostrand.
Kovačič, M., Lavrač, N., Grobelnik, M., Zupančič, D., & Mladenič, D. (1992). Stochastic search in inductive logic programming. In *Proceedings of the European Conference on Artificial Intelligence*. (pp. 444–445).
Muggleton, S. (1995). Inverse Entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming 13*(3-4), 245–286.
Pompe, U., Kononenko, & I., Makše T. (1996). An application of ILP in a musical database: Learning to compose the two-voice counterpoint. *Proceedings of the MLnet Familiarization Workshop on Data Mining with Inductive Logic Programming*. (pp. 1–11).
Pompe, U., Kovačič, M., & Kononenko I. (1993). SFOIL: Stochastic approach to inductive logic programming. *Proceedings of the 2nd slovenian conference on electrotechnical engineering and computer science*.
Selman, B., Levesque, H. J., & Mitchell D. (1992). A new method for solving hard satisfiability problems. In *Proceedings of the tenth national conference on artificial intelligence* (pp. 440–446). AAAI Press.
Serrurier, M., Prade, H., & Richard G. (2004). A simulated annealing framework for ILP. In *Proceedings of the 14th international conference.* (pp. 289–304) Springer.
Srinivasan, A., Muggleton, S., Sternberg, M. J. E., & King, R. D. (1996). Theories for mutagenicity: A study in first-order and feature-based induction'. *Artificial Intelligence*, *85*(1–2), 277–299.
Trefethen, N. (1998). Maxims about numerical mathematics, computers, science, and life. *SIAM News*.
Železný, F., Srinivasan, A., & Page, D. (2003). Lattice-search runtime distributions may be heavy-tailed. In *Proceedings of the 12th international conference on inductive logic programming* (pp. 333–345).
Zilberstein, S. (1998). Satisficing and bounded optimality. In *AAAI Spring symposium on satisficing models.*