Taming the Complexity of Inductive Logic Programming

SOFSEM 2010

Filip Železný and Ondřej Kuželka

Czech Technical University in Prague

February 2, 2010

Table of contents

1. Inductive Logic Programming
2. Taming Clause Evaluation Complexity
3. Taming Clause Search Complexity
4. Conclusions

Inductive Logic Programming

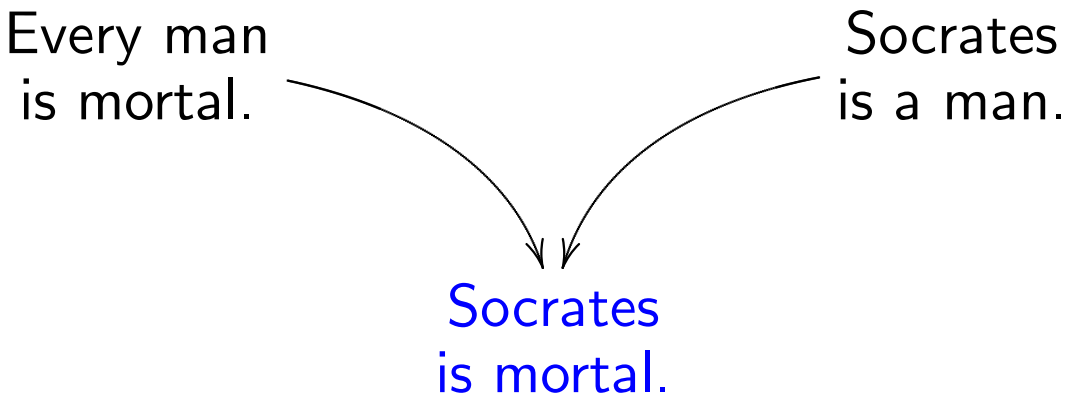
a no-tears introduction

Deduction

Every man
is mortal.

Socrates
is a man.

Deduction

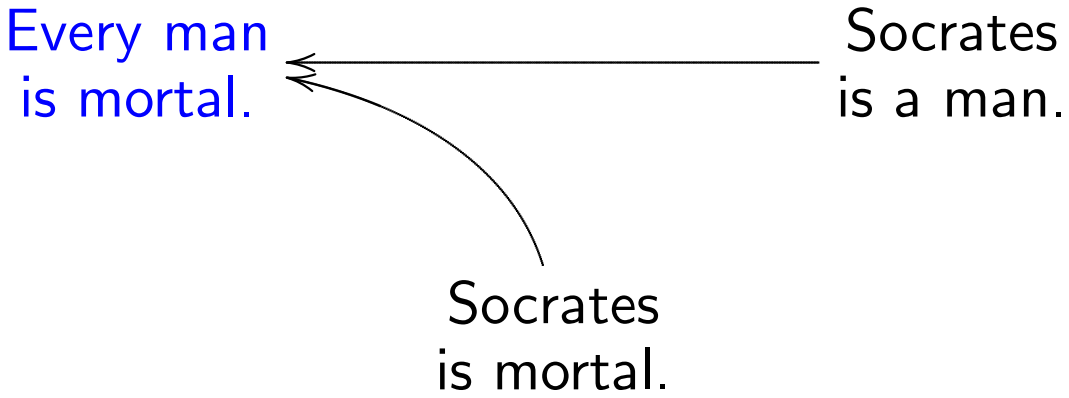


Induction

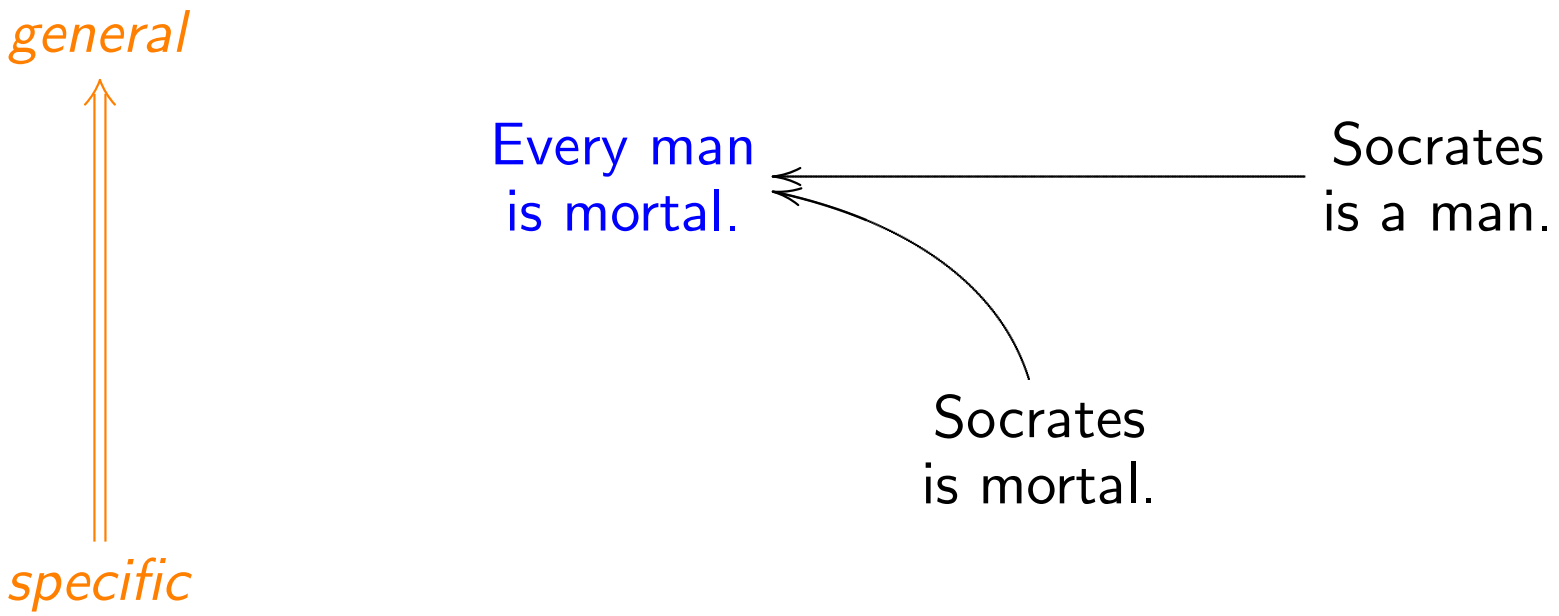
Socrates
is a man.

Socrates
is mortal.

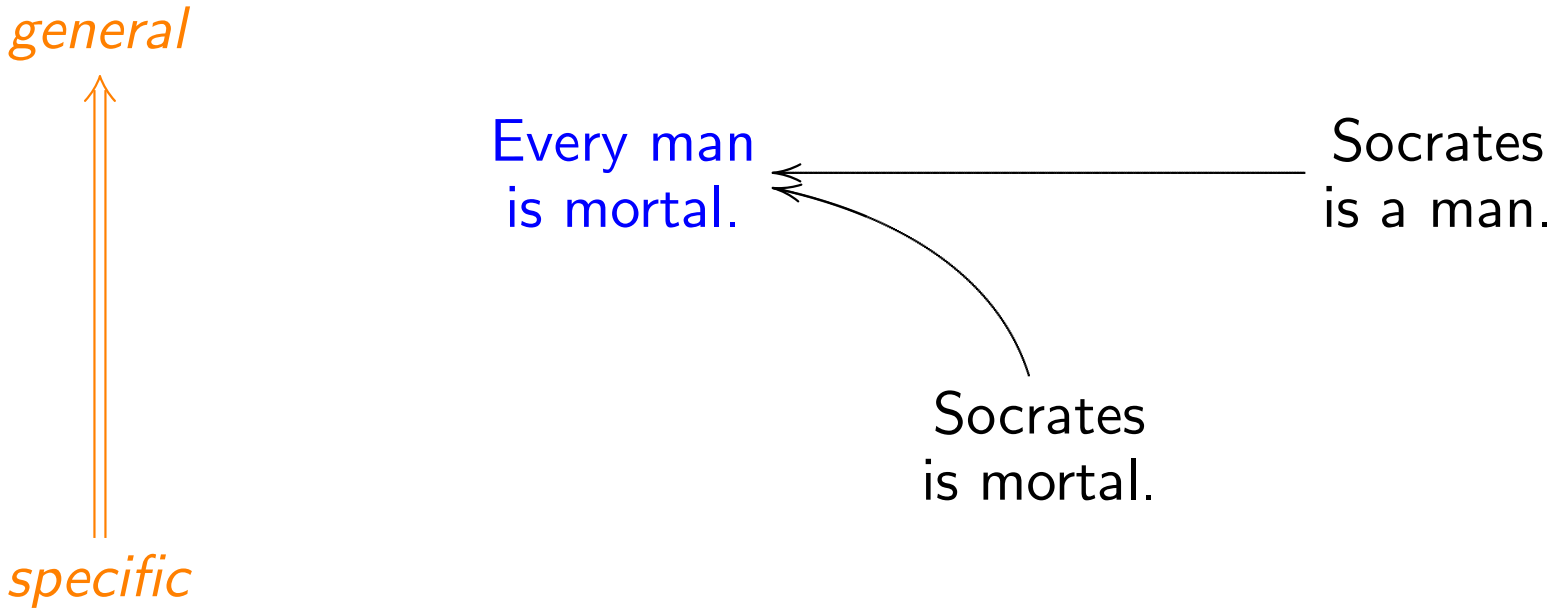
Induction



Induction


general
specific
Every man is mortal.
Socrates is a man.
Socrates is mortal.

Induction



Induction is not sound.

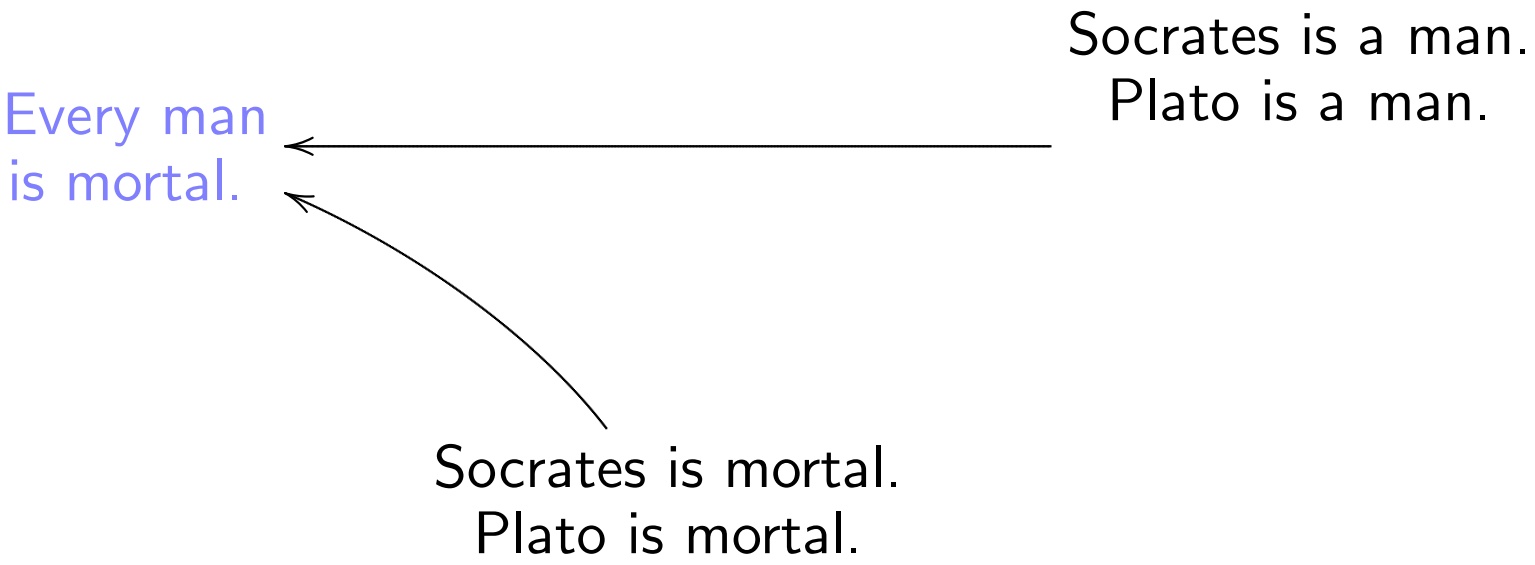
Justification

Socrates is a man.

Every man is mortal.

Socrates is mortal.

Justification



Justification

Every man is mortal.

Socrates is a man.
Plato is a man.
Aristotle is a man.

Socrates is mortal.
Plato is mortal.
Aristotle is mortal.

Justification

Every man is mortal.

Socrates is a man.
Plato is a man.
Aristotle is a man.
⋮

Socrates is mortal.
Plato is mortal.
Aristotle is mortal.
⋮

Induction

Induction =

Generalization + Justification

Induction

Induction =

Generalization + Justification

Logic Statistics

Generalization in Logic

Logic provides

- a **language**, e.g. first-order clauses

$$C = \text{man}(\text{Socrates}) \rightarrow \text{man}(\text{Socrates})$$
$$D = \forall x\ \text{man}(x) \rightarrow \text{mortal}(x)$$

- a **generality relation**, e.g. entailment

$$D \models C$$

A Machine Learning View

- through induction, one can **learn** general **concepts**, e.g.

$$\forall x \ \mathrm{man}(x) \rightarrow \mathrm{mortal}(x)$$

- from numerous specific **examples**, e.g.

$$\mathrm{man}(\mathrm{Socrates}) \rightarrow \mathrm{man}(\mathrm{Socrates})$$

Counterexamples

The intended concept

$$\forall x \ \mathrm{man}(x) \rightarrow \mathrm{mortal}(x)$$

is justified by examples but so is the concept

$$C = \forall x \ \mathrm{mortal}(x)$$

C is overly general:

$$C \models \mathrm{mortal}(\text{Plato's legacy})$$

As a prevention, an induction task also requires counterexamples (or: negative examples) the concept should not entail.

Inductive Logic Programming: Definition

Muggleton, Brazdil et al., early 90's

The Normal ILP Task

Given

- E^+ a set of positive examples (clauses)
- E^- a set of negative examples (clauses)

Inductive Logic Programming: Definition

Muggleton, Brazdil et al., early 90's

The Normal ILP Task

Given

- E^+ a set of positive examples (clauses)
- E^- a set of negative examples (clauses)
- B *background knowledge* (set of clauses) such that
  - $B \models e$ for **no** $e \in E^+ \cup E^-$

Inductive Logic Programming: Definition

Muggleton, Brazdil et al., early 90's

The Normal ILP Task

Given

- E^+ a set of positive examples (clauses)
- E^- a set of negative examples (clauses)
- B *background knowledge* (set of clauses) such that
  - $B \models e$ for **no** $e \in E^+ \cup E^-$

Find

- H *hypothesis* (set of clauses) that is consistent, i.e.
  - $H \cup B \models e$ for **no** $e \in E^-$

Inductive Logic Programming: Definition

Muggleton, Brazdil et al., early 90's

The Normal ILP Task

Given

- E^+ a set of positive examples (clauses)
- E^- a set of negative examples (clauses)
- B *background knowledge* (set of clauses) such that
  - $B \models e$ for **no** $e \in E^+ \cup E^-$

Find

- H *hypothesis* (set of clauses) that is consistent, i.e.
  - $H \cup B \models e$ for **no** $e \in E^-$
  - $H \cup B \models e$ for **all** $e \in E^+$

Inductive Logic Programming: Definition

Muggleton, Brazdil et al., early 90's

The Normal ILP Task

Given

- E^+ a set of positive examples (clauses)
- E^- a set of negative examples (clauses)
- B *background knowledge* (set of clauses) such that
  - $B \models e$ for **no** $e \in E^+ \cup E^-$

Find

- H *hypothesis* (set of clauses) that is consistent, i.e.
  - $H \cup B \models e$ for **no** $e \in E^-$
  - $H \cup B \models e$ for **all** $e \in E^+$

Note: all sets finite, set of clauses = *clausal theory*

Examples

A Family Example

$E^+ = \{\text{grandson}(\text{Bart}, \text{Abe}), \text{grandson}(\text{Stewie}, \text{Carter})\}$

$E^- = \{\text{grandson}(\text{Lisa}, \text{Abe}), \text{grandson}(\text{Homer}, \text{Peter})\}$

Examples

A Family Example

$$E^+ = \{\text{grandson}(\text{Bart}, \text{Abe}), \text{grandson}(\text{Stewie}, \text{Carter})\}$$
$$E^- = \{\text{grandson}(\text{Lisa}, \text{Abe}), \text{grandson}(\text{Homer}, \text{Peter})\}$$
$$B = \{\text{son}(\text{Bart}, \text{Homer}), \text{son}(\text{Stewie}, \text{Lois}), \text{son}(\text{Homer}, \text{Abe}), \text{daughter}(\text{Lois}, \text{Carter})\}$$

Examples

A Family Example

$$
\begin{aligned}
E^+ &= \{\text{grandson}(\text{Bart}, \text{Abe}), \text{grandson}(\text{Stewie}, \text{Carter})\} \\
E^- &= \{\text{grandson}(\text{Lisa}, \text{Abe}), \text{grandson}(\text{Homer}, \text{Peter})\} \\
B &= \{\text{son}(\text{Bart}, \text{Homer}), \text{son}(\text{Stewie}, \text{Lois}), \\
&\quad \text{son}(\text{Homer}, \text{Abe}), \text{daughter}(\text{Lois}, \text{Carter})\} \\
H &= \{\text{son}(x, y) \wedge \text{son}(y, z) \rightarrow \text{grandson}(x, z) \\
&\quad \text{son}(x, y) \wedge \text{daughter}(y, z) \rightarrow \text{grandson}(x, z)
\end{aligned}
$$

Note: '$\forall x \forall y \forall z$' implicit

Examples (cont'd)

A Logic Programming Example

$$E^+ = \{\texttt{member(a,[a,b,c]), member(3,[1,2,3,4]). ...}\}$$
$$E^- = \{\texttt{member(2,[a,b]), member(b,[1,2,3,4,6]). ...}\}$$

Examples (cont'd)

A Logic Programming Example

E^+ = {member(a,[a,b,c]), member(3,[1,2,3,4]). ... }
E^- = {member(2,[a,b]), member(b,[1,2,3,4,6]). ... }
B = {head([H|_],H). tail([_|T],T).}

Examples (cont'd)

A Logic Programming Example

E^+ = {member(a,[a,b,c]), member(3,[1,2,3,4]). ...}

E^- = {member(2,[a,b]), member(b,[1,2,3,4,6]). ...}

B = {head([H|_],H). tail([_|T],T).}

H = {

```
member(A,B) :- head(B,A).
member(A,B) :- tail(B,T), member(A,T).
```

}

A Logic Programming View

ILP =

Logic Programming $\cap$ Machine Learning

A Logic Programming View

ILP =

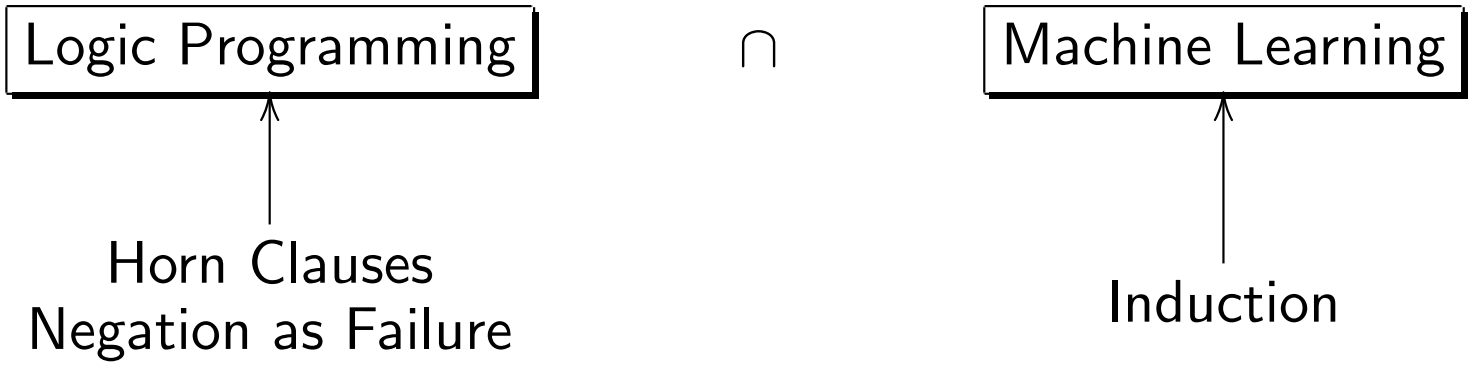
Logic Programming ∩ Machine Learning

Horn Clauses
Negation as Failure

Induction

A Logic Programming View

ILP =



- ILP as a means for automated synthesis of logic programs from examples
- a popular view in the 90' (not any longer)

Examples (cont'd)

Mortality over again

$$
\begin{aligned}
E^+ =& \{\text{male(Socrates)} \rightarrow \text{mortal(Socrates)}, \\
& \text{male(Plato)} \rightarrow \text{mortal(Plato)}\} \\
E^- =& \{\text{mortal(Socrates' legacy)}, \text{mortal(Plato's legacy)}\} \\
B =& \{\text{man}(x) \rightarrow \text{mortal}(x)\} \\
H =&
\end{aligned}
$$

Examples (cont'd)

Mortality over again

$$
\begin{aligned}
E^+ =& \{\text{male(Socrates)} \rightarrow \text{mortal(Socrates)}, \\
& \text{male(Plato)} \rightarrow \text{mortal(Plato)}\} \\
E^- =& \{\text{mortal(Socrates' legacy)}, \text{mortal(Plato's legacy)}\} \\
B =& \{\text{man}(x) \rightarrow \text{mortal}(x)\} \\
H =& \{\text{male}(x) \rightarrow \text{man}(x)\}
\end{aligned}
$$

Examples (cont'd)

Mortality over again

$$
\begin{aligned}
E^+ =& \{\text{male(Socrates)} \rightarrow \text{mortal(Socrates)}, \\
& \text{male(Plato)} \rightarrow \text{mortal(Plato)}\} \\
E^- =& \{\text{mortal(Socrates' legacy)}, \text{mortal(Plato's legacy)}\} \\
B =& \{\text{man}(x) \rightarrow \text{mortal}(x)\} \\
H =& \{\text{male}(x) \rightarrow \text{man}(x)\}
\end{aligned}
$$

- **mortal** predicate not in the induced concept description!
- indicates generality ($\Rightarrow$ difficulty) of the normal ILP task

Solving the ILP task

A general scheme

1. Explore some predefined hypothesis space $\mathcal{H}$
2. Output the first $H \in \mathcal{H}$ consistent with E^+, E^-

- Here $H \cup B \models e$ must be checked for all $H \in \mathcal{H}$ and $e \in E^+ \cup E^-$
- First blocker: entailment $\models$ undecidable
- Go humble: let H **be a single clause and** B **empty**

Solving the ILP task

A general scheme

1. Explore some predefined hypothesis space $\mathcal{H}$
2. Output the first $H \in \mathcal{H}$ consistent with E^+, E^-

- Here $H \cup B \models e$ must be checked for all $H \in \mathcal{H}$ and $e \in E^+ \cup E^-$
- First blocker: entailment $\models$ undecidable
- Go humble: let H **be a single clause and** B **empty**
- Nice try but $H \models e$ still undecidable! [Marcinowski, Pacholski '92]

Subsumption

- Remedy: use subsumption instead of entailment

Subsumption between clauses H and e

$H \preceq_\theta e$ if, for some substitution θ, e **contains all literals of** $H\theta$

Subsumption

- Remedy: use subsumption instead of entailment

Subsumption between clauses H and e

$H \preceq_\theta e$ if, for some substitution θ, e **contains all literals of** $H\theta$

- Example:

$$\begin{aligned} & \text{mortal}(x) \\ \preceq_\theta\ & \text{man}(x) \rightarrow \text{mortal}(x) \\ \preceq_\theta\ & \text{man}(\text{Sokrates}) \rightarrow \text{mortal}(\text{Sokrates}) \end{aligned}$$

Subsumption

- Remedy: use subsumption instead of entailment

Subsumption between clauses H and e

$H \preceq_\theta e$ if, for some substitution θ, e **contains all literals of** $H\theta$

- Example:

$$\begin{aligned} & \text{mortal}(x) \\ \preceq_\theta\ & \text{man}(x) \rightarrow \text{mortal}(x) \\ \preceq_\theta\ & \text{man}(\text{Sokrates}) \rightarrow \text{mortal}(\text{Sokrates}) \end{aligned}$$

- Subsumption is only an approximation for entailment
  - (equivalent for non-self-resolving clauses)

ILP task: Decidability

Summary: decidability guaranteed if simultaneously

- H is a single clause
  - workaround: clausal theories learned in a greedy clause-by-clause way
  - using a *covering strategy* known from good-old-fashioned rule learning in AI

ILP task: Decidability

Summary: decidability guaranteed if simultaneously

- H is a single clause
  - workaround: clausal theories learned in a greedy clause-by-clause way
  - using a *covering strategy* known from good-old-fashioned rule learning in AI
- H is not self-resolving
  - excludes synthesis of logic programs with recursion
  - e.g. $\text{member}(x, y) \leftarrow \text{tail}(y, t) \wedge \text{member}(x, t)$

ILP task: Decidability

Summary: decidability guaranteed if simultaneously

- H is a single clause
  - workaround: clausal theories learned in a greedy clause-by-clause way
  - using a *covering strategy* known from good-old-fashioned rule learning in AI
- H is not self-resolving
  - excludes synthesis of logic programs with recursion
  - e.g. $\text{member}(x, y) \leftarrow \text{tail}(y, t) \wedge \text{member}(x, t)$
- B is empty
  - if B contains only ground facts, task can be *reformulated* to one with B empty
  - In general, task can be *approximately* reformulated to one with B empty

ILP task: Complexity

The **ILP bounded consistency problem** [Gottlob 1999]

- Assumptions:
  - the normal ILP task
  - (cca) the decidability conditions stated earlier
  - H has at most $poly(E^+ \cup E^-)$ literals
- Result: the problem is NP^{NP}-complete
- Two oracles would be needed for tractability
  - To guess a consistent hypothesis
  - To guess θ in checking $H \preceq_\theta e$

ILP task: Complexity

The **ILP template consistency problem** [Gottlob 1999]

- Assumptions:
  - the normal ILP task
  - (cca) the decidability conditions stated earlier
  - $H = T\vartheta$, T is a given template
- Result: the problem is NP^{NP}-complete
- Two oracles would be needed for tractability
  - To guess ϑ
  - To guess θ in checking $H \preceq_\theta e$

ILP task: Complexity (cont'd)

Summary:

Sources of ILP Complexity

1. The Complexity of Clause Evaluation
2. The Complexity of Clause Search

ILP task: Complexity (cont'd)

Summary:

Sources of ILP Complexity

1. The Complexity of Clause Evaluation
2. The Complexity of Clause Search

Disclaimer for machine learners:

- solving the ILP task efficiently is not a sufficient condition for PAC-learning of clauses

Taming Clause Evaluation Complexity

Conquering Subsumption Complexity

- Deciding subsumption is an NP-complete problem

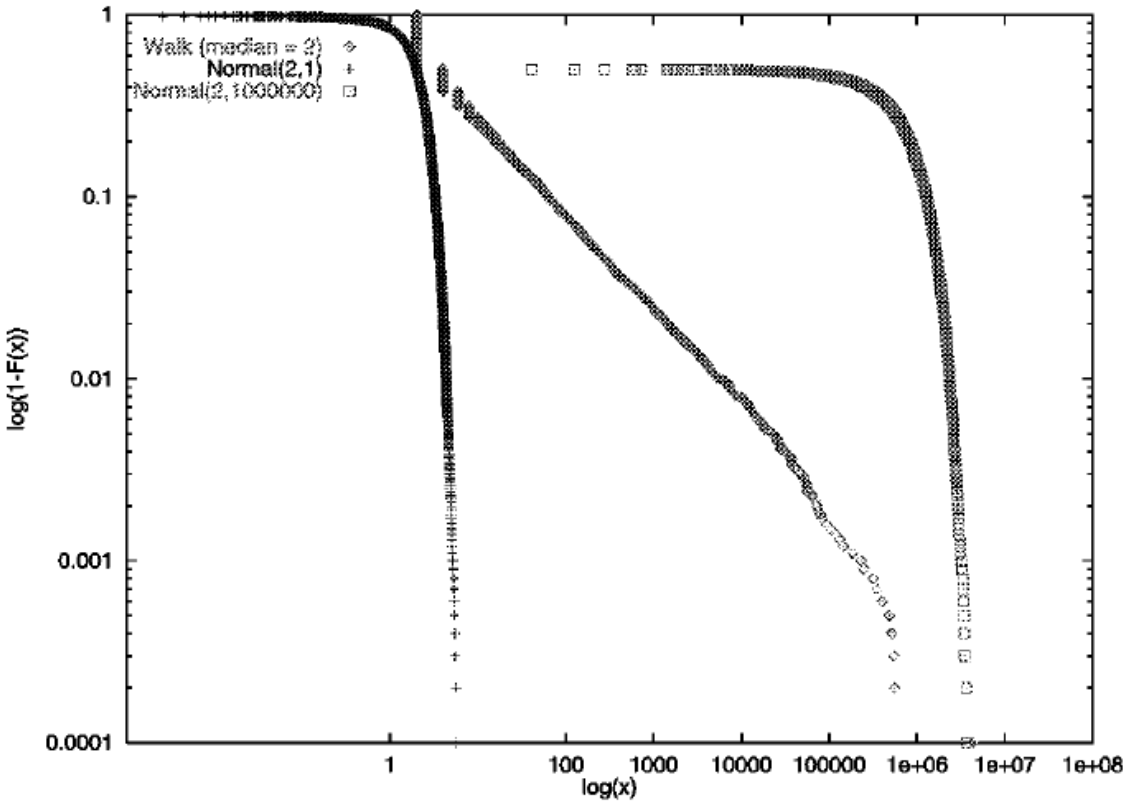
Previous work to enhance subsumption testers

- Incomplete randomized algorithms
  - Stochastic Matching [Sebag et al 1997]
  - Table based methods [Arias 2007]
- Complete algorithm Django [Maloberti et al 2004]
  - Reformulation as a constraint satisfaction problem
  - Pro: Exploits CSP heuristics, dramatic speed-up in most cases
  - Exhibits heavy tails

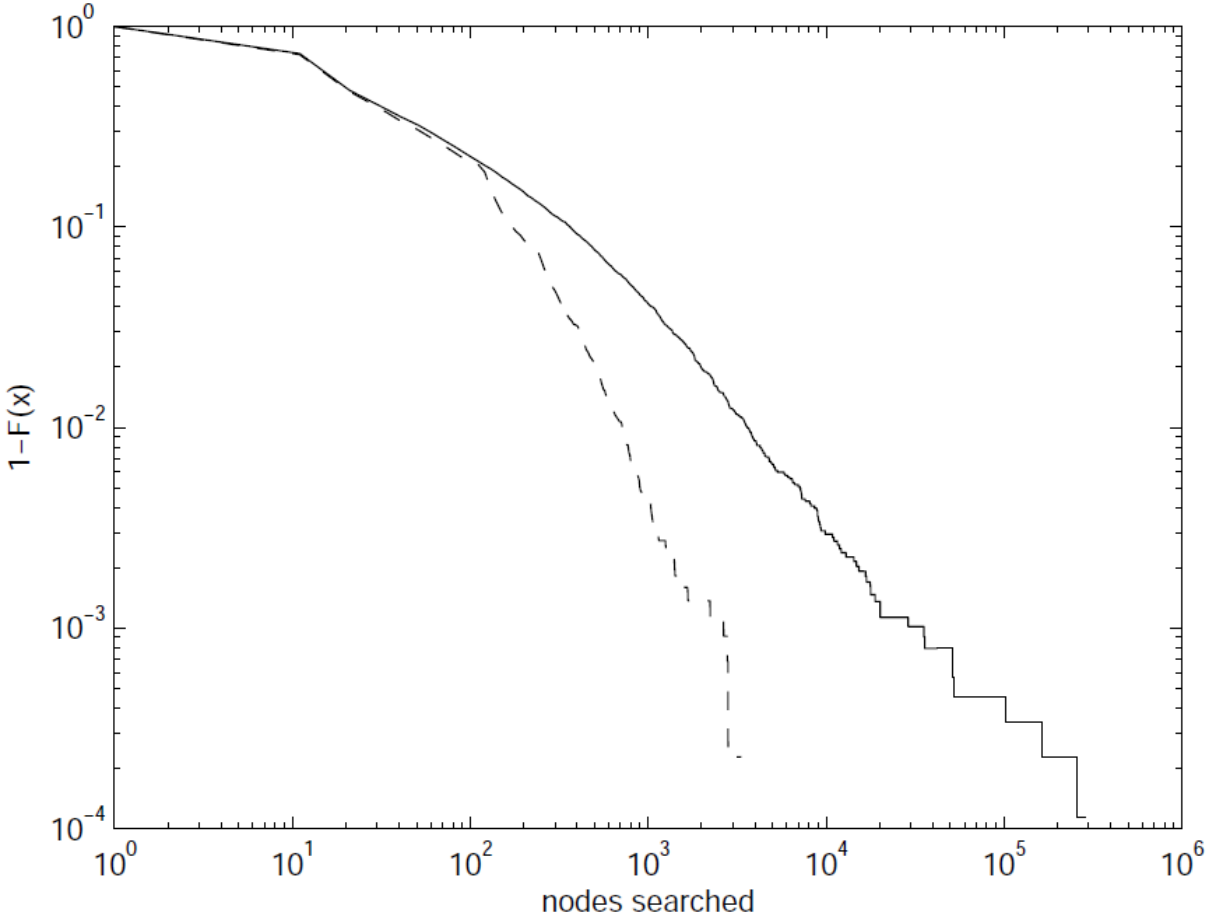
Heavy Tailed Runtime Distributions

Heavy tailed phenomenon in combinatorial problems

[Gomes Selman et al, 1998, 2000, ...]



Subsumption Runtime Distributions

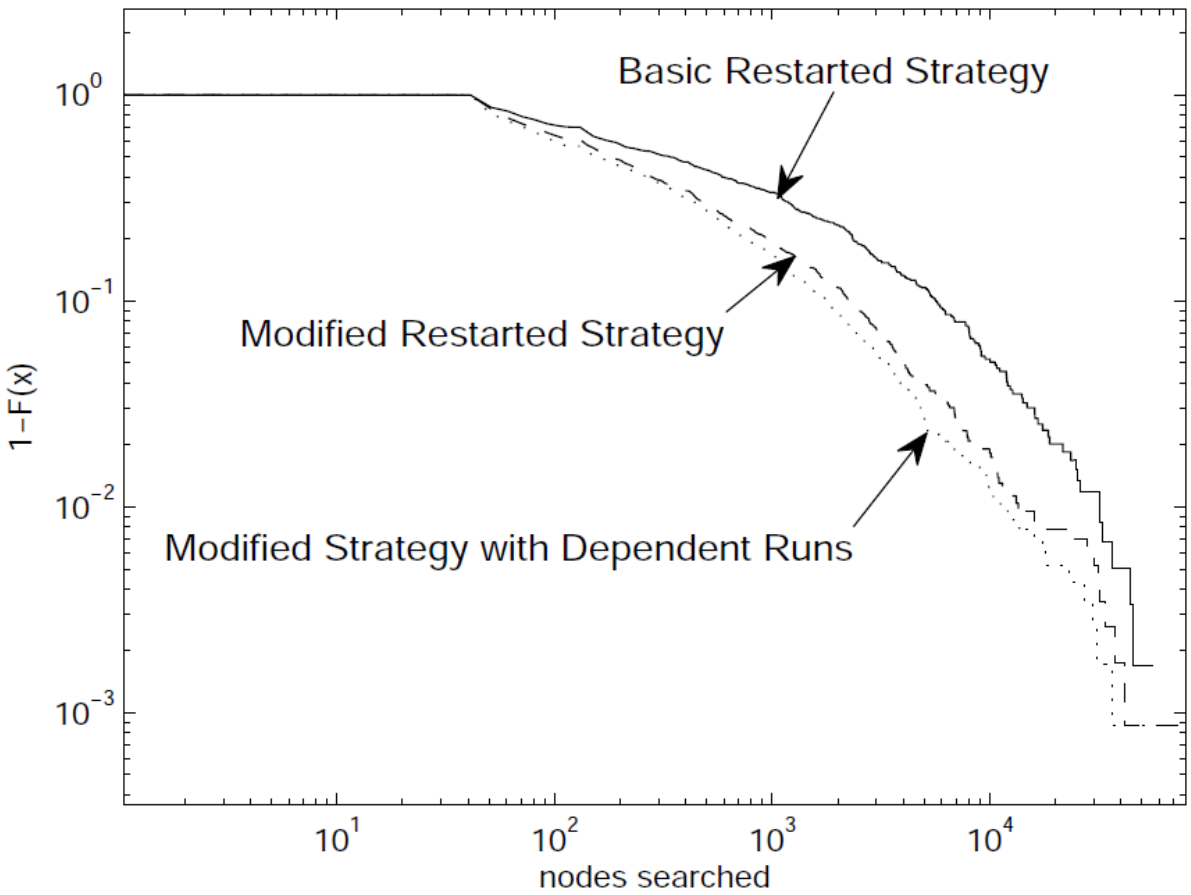


A Restarted Subsumption Tester

- ReSumER: A Restarted Subsumption Tester

  [Kuzelka, Zelezny - Fundamenta Informaticae '08]
- A restarted randomized CSP procedure
- Growing cutoff value $10e^n + 100$ in n-th try
- A complete algorithm (finite search space)

ReSumEr Runtime Distributions



A Restarted Coverage Estimator

- ReCovEr: A Restarted Coverage Estimator

  [Kuzelka, Zelezny - ICML '08]

- Motto: *to solve a problem, do not solve a harder problem as an intermediate step* [Vapnik]

A Restarted Coverage Estimator (cont'd)

- In 'real life' ILP, **fully consistent** hypotheses rarely available
- One looks for a hypothesis with the **best fit** on data
- E.g.

$$Fit = P - N$$

P : number of positives Subsumed
N : number of negatives subsumed

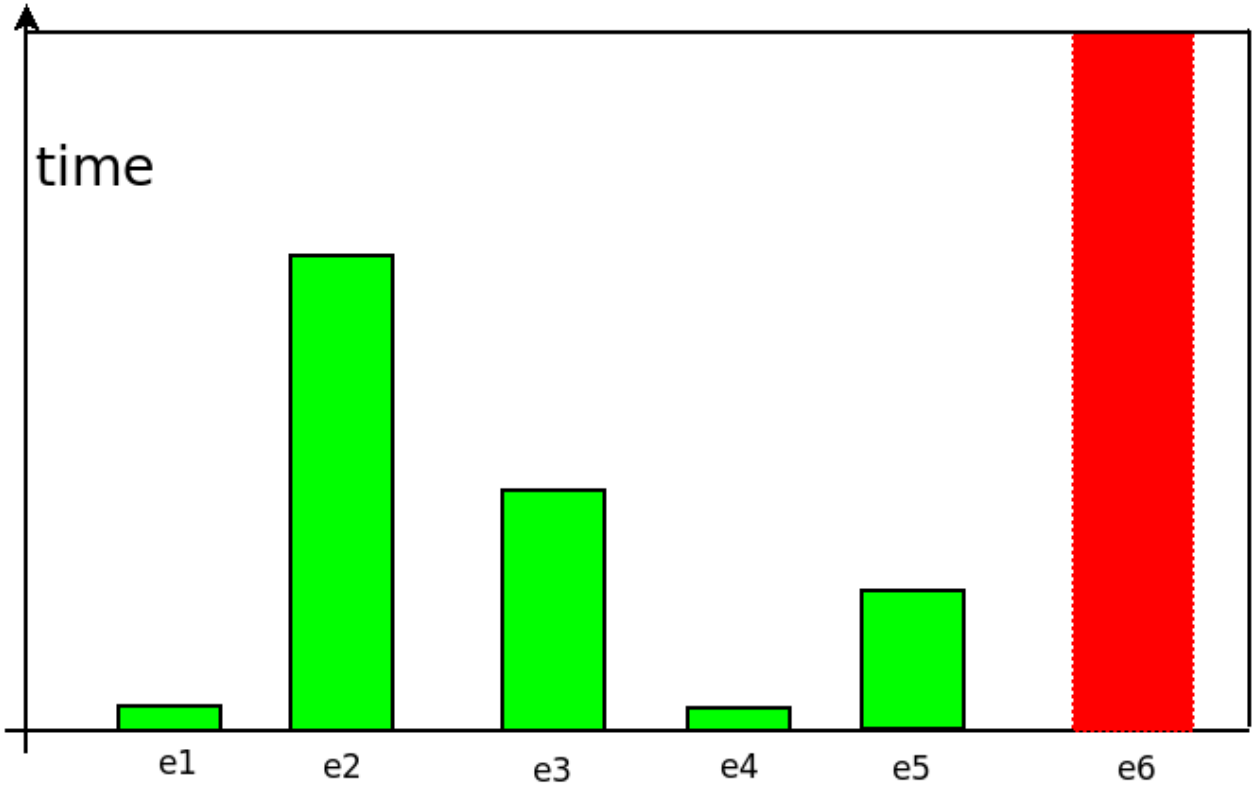
A Restarted Coverage Estimator (cont'd)

- In 'real life' ILP, **fully consistent** hypotheses rarely available
- One looks for a hypothesis with the **best fit** on data
- E.g.

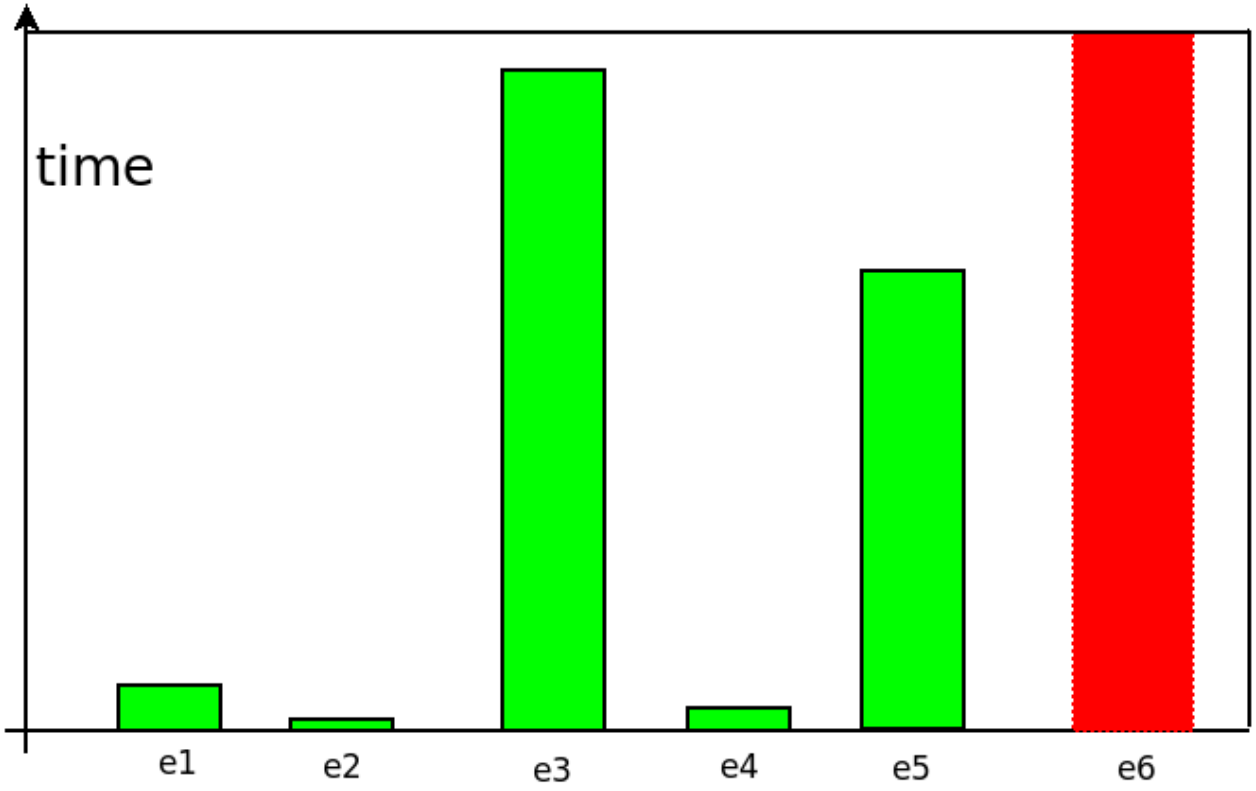
$$Fit = P - N$$

  P : number of positives Subsumed
  N : number of negatives subsumed
- Do not care to decide **subsumption for each example**
- Just need to know P and N
- Estimates will do
  - No *loss of correctness*: examples come from random sampling anyway!

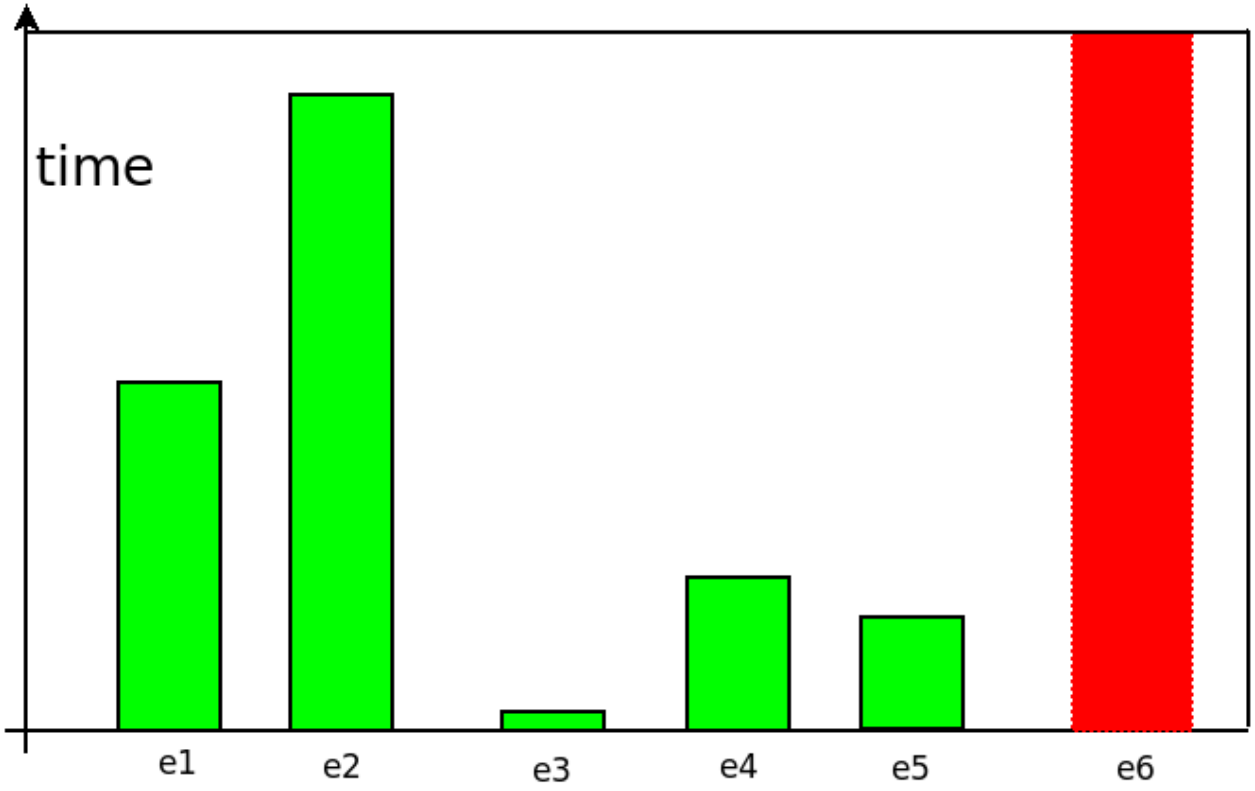
ReCovEr Principle



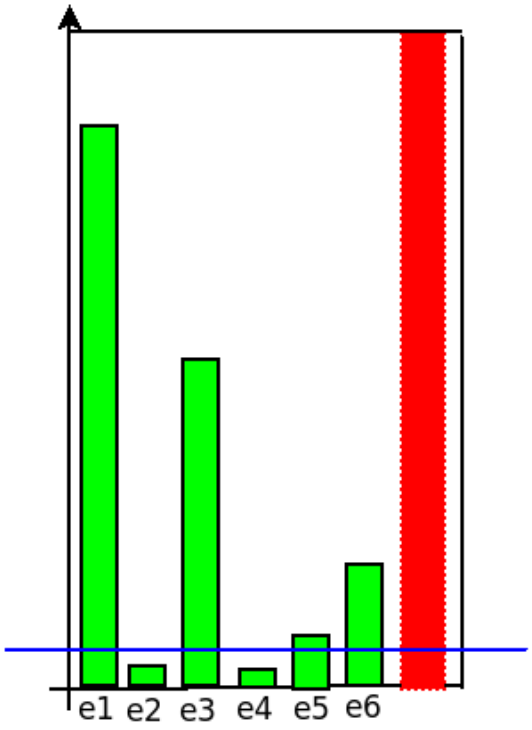
ReCovEr Principle



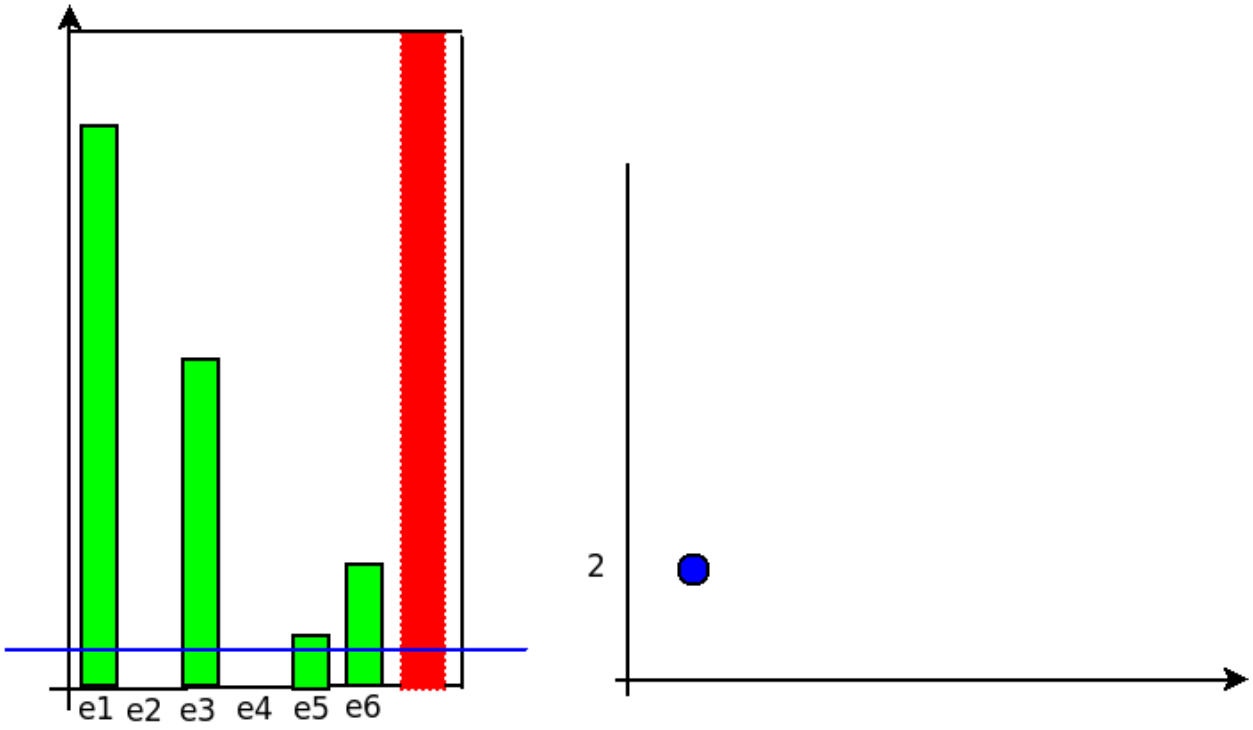
ReCovEr Principle



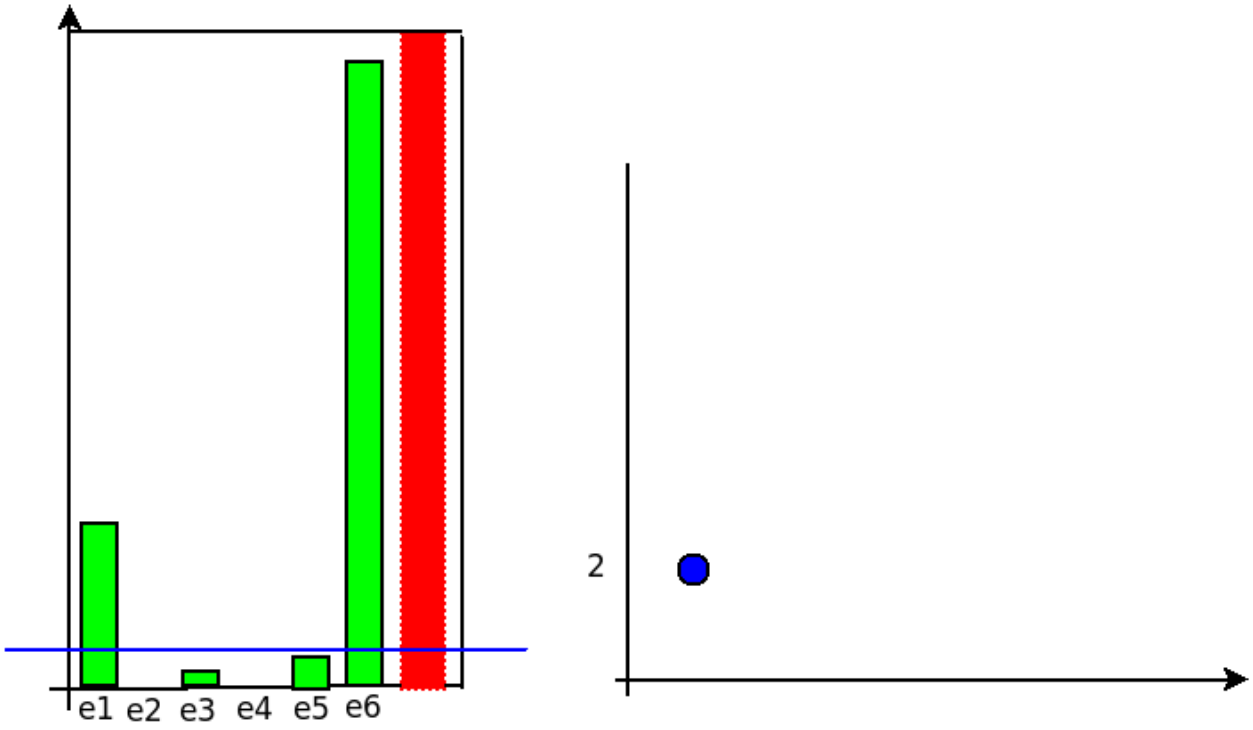
ReCovEr Principle



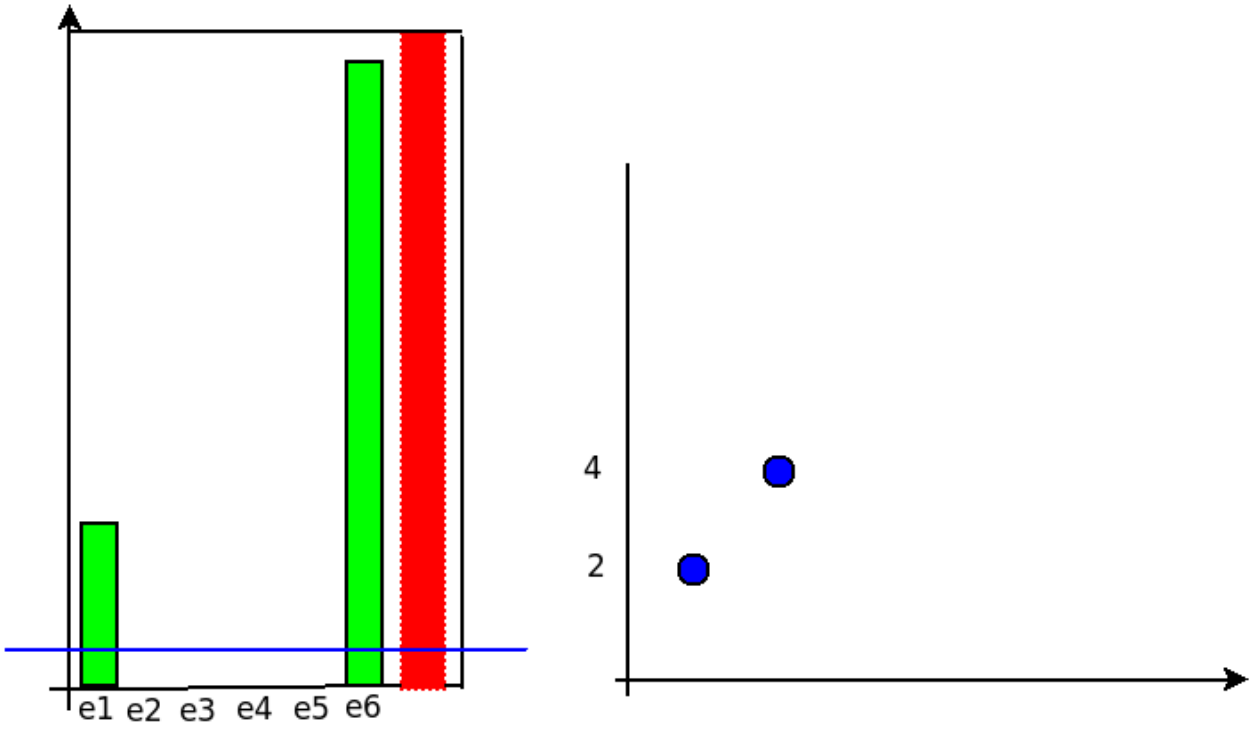
ReCovEr Principle



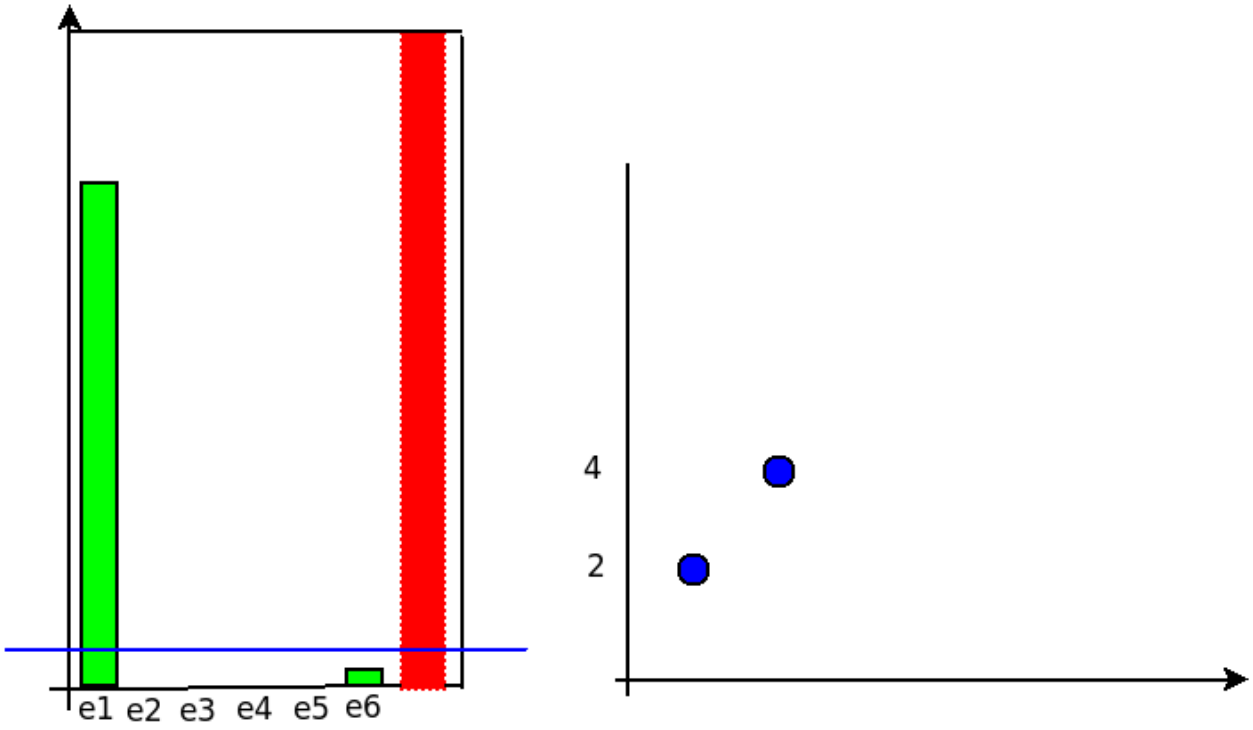
ReCovEr Principle



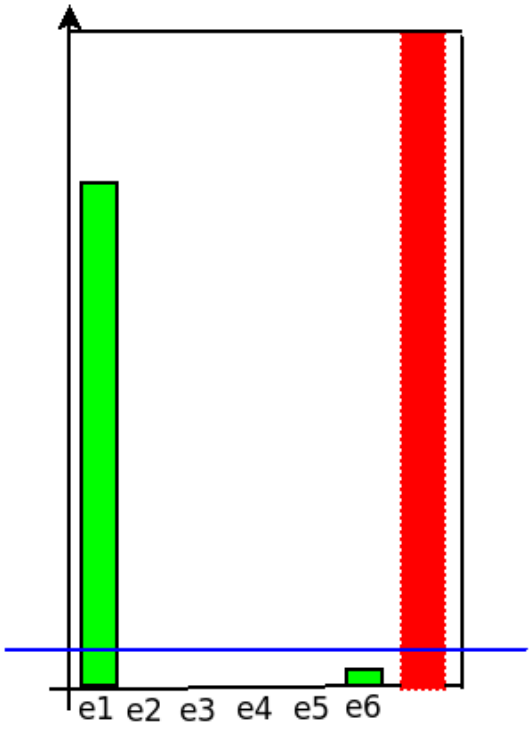
ReCovEr Principle

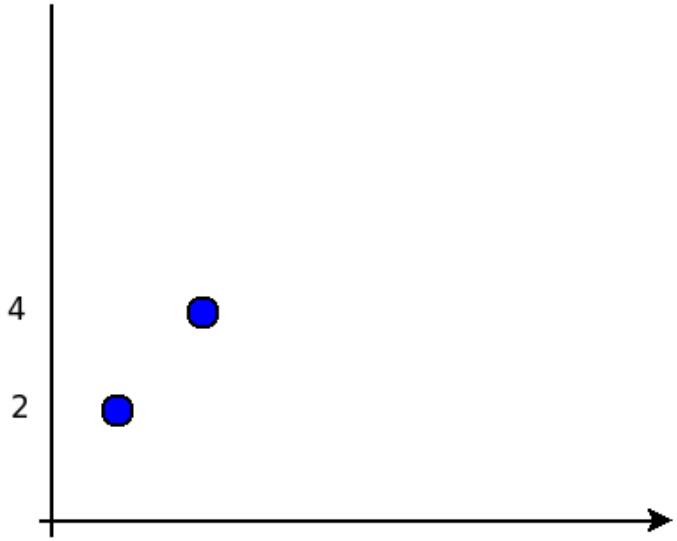


ReCovEr Principle

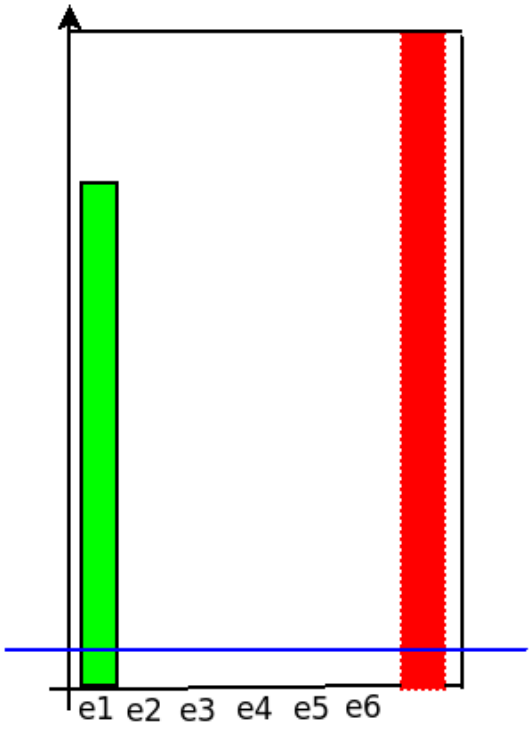


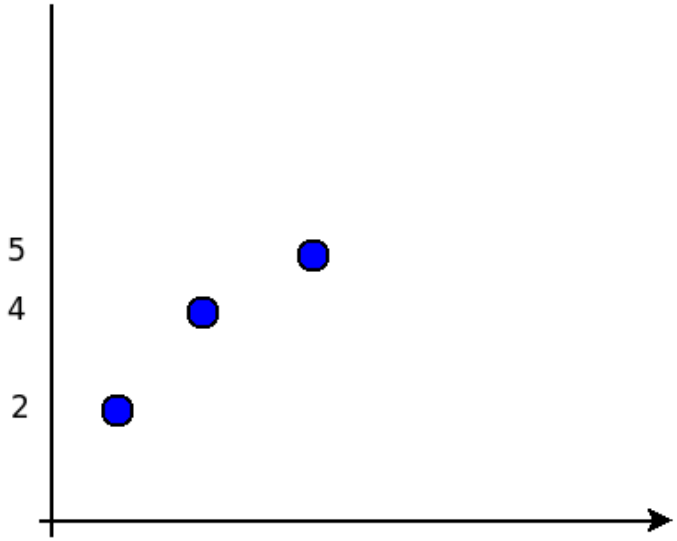
ReCovEr Principle



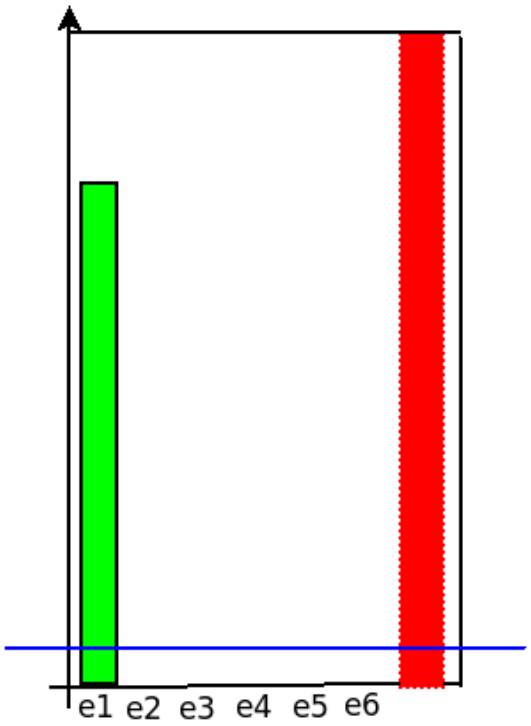


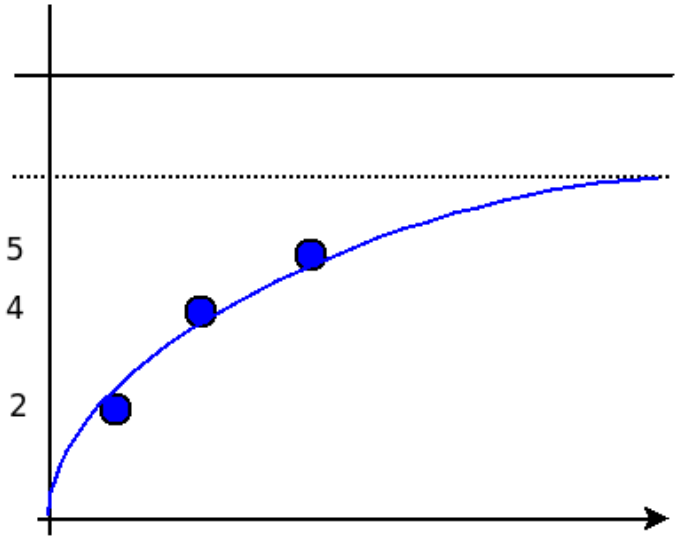
ReCovEr Principle



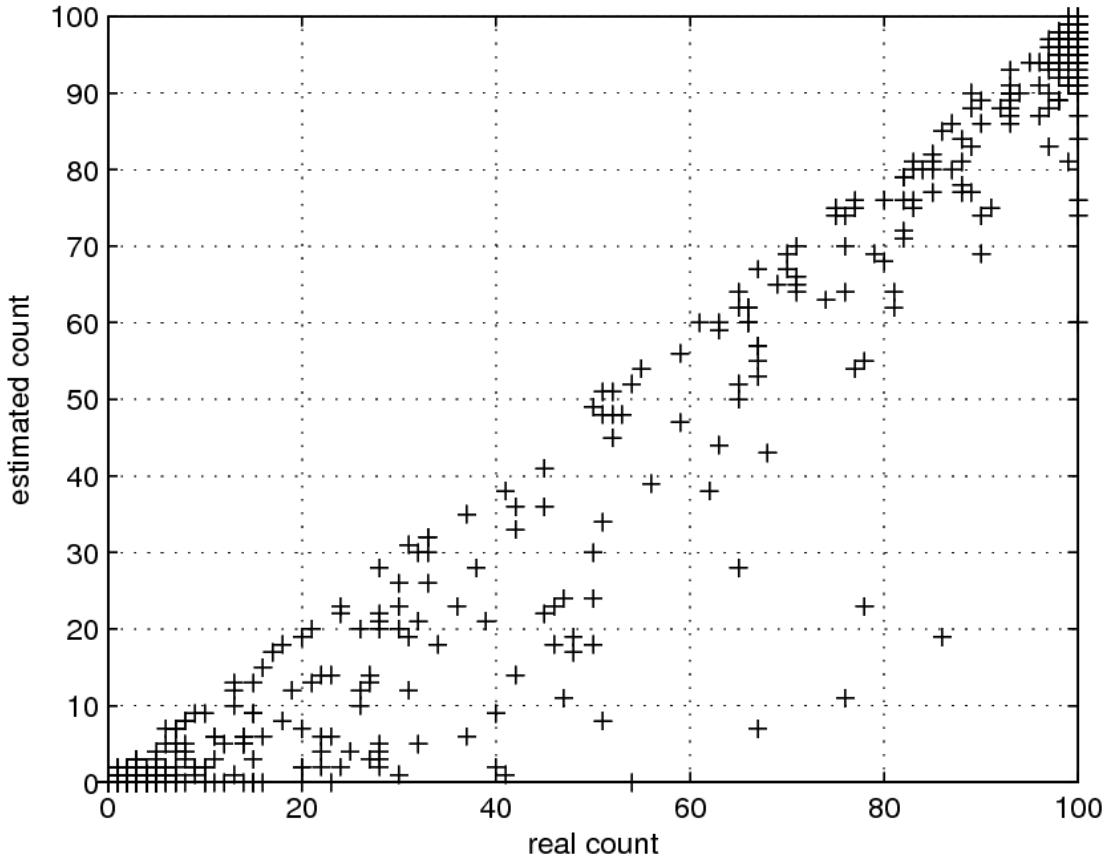


ReCovEr Principle



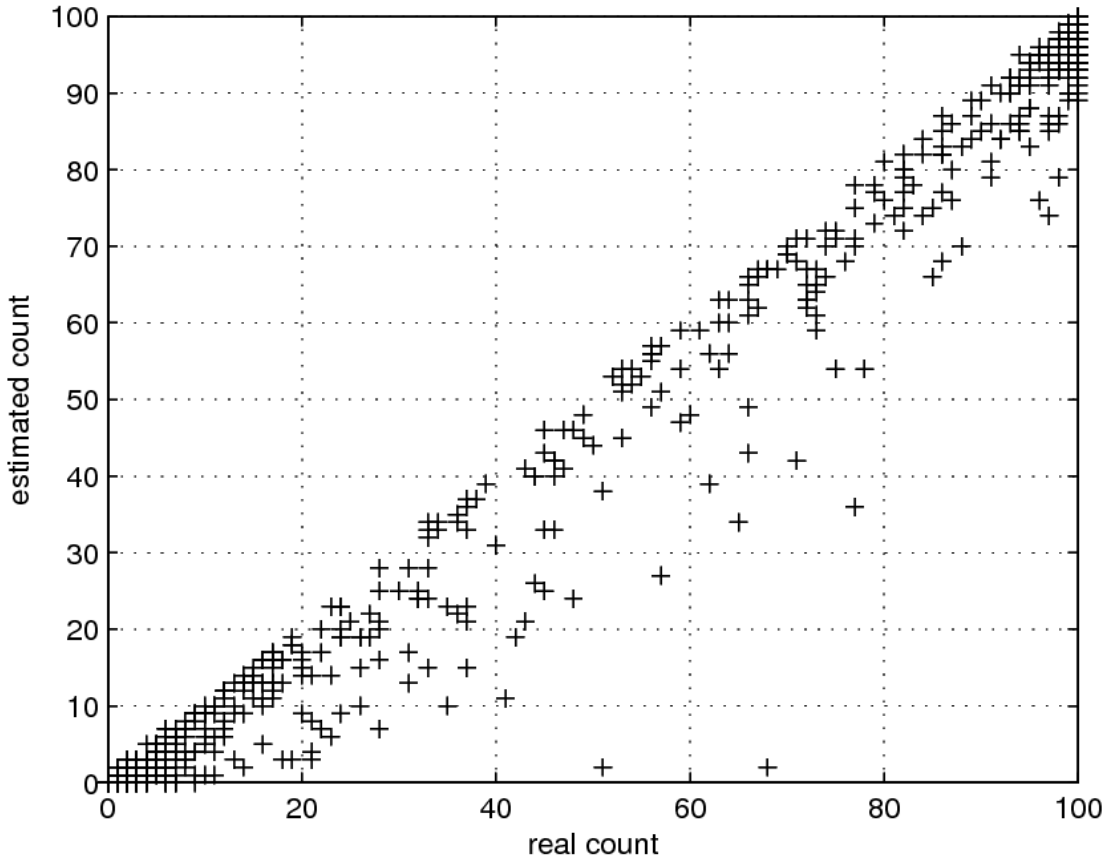


ReCovEr Performance



cutoff 100 backtracks

ReCovEr Performance



cutoff 200 backtracks

ReCovEr Performance vs. Django

Algorithm	Avg. Time [s]
ReCovEr, $R = 100$	6.7
ReCovEr, $R = 200$	12.5
Django	483.2

Randomized Restarted Strategies for Clause Search

- A Mega Study (100's of computers X days) of 5 search strategies in 3 induction problems

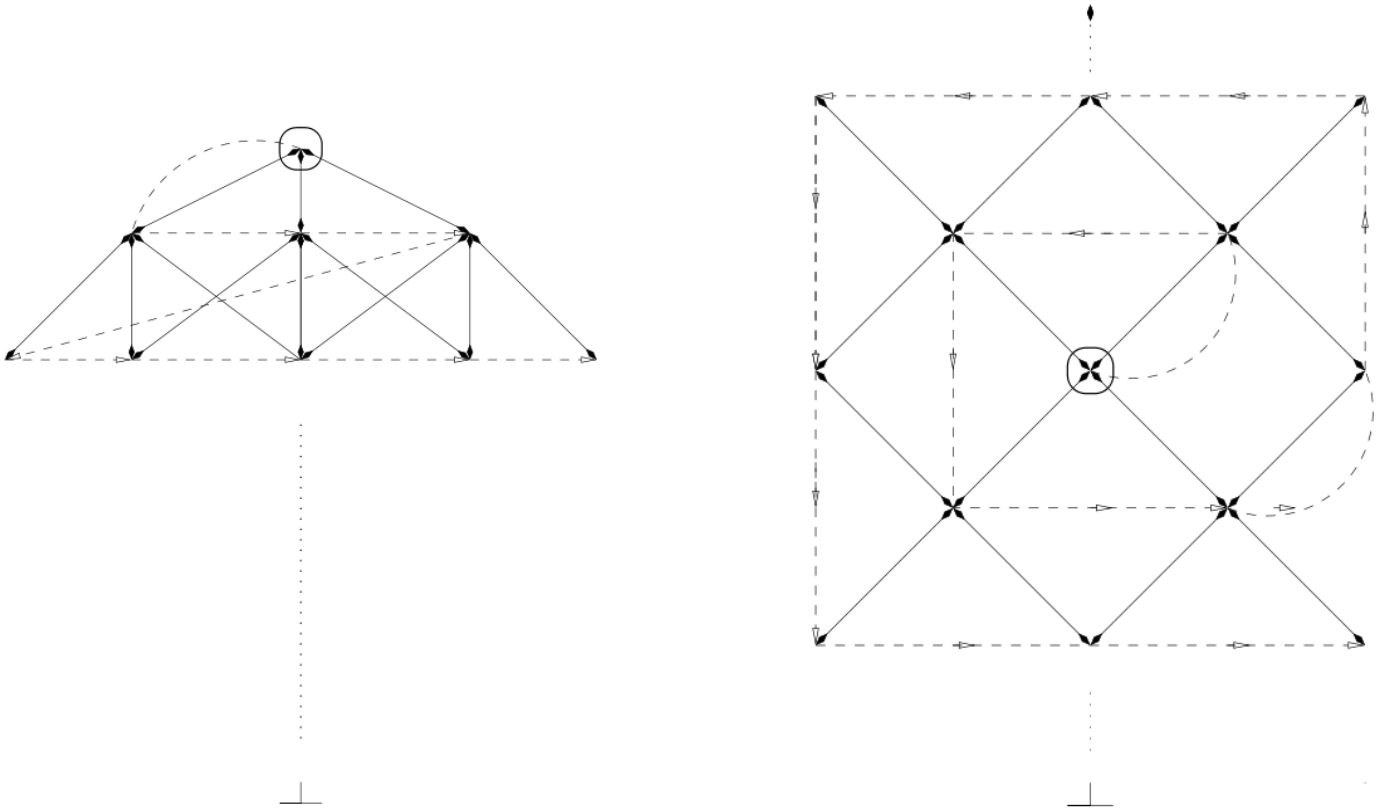
  [Zelezny, Srinivasan, Page - Machine Learning 2006]

Strategy → ↓ *Step*	DTD	RTD	RRR	GSAT	WSAT
Saturant selection	D	R	R	R	R
Start clause selection	D	D	R	R	R
Update active list	C	C	C	G	G
Next clause selection	D	R	D	D	R
Pruning	Y	Y	N	N	N
Refinement	U	U	B	B	B

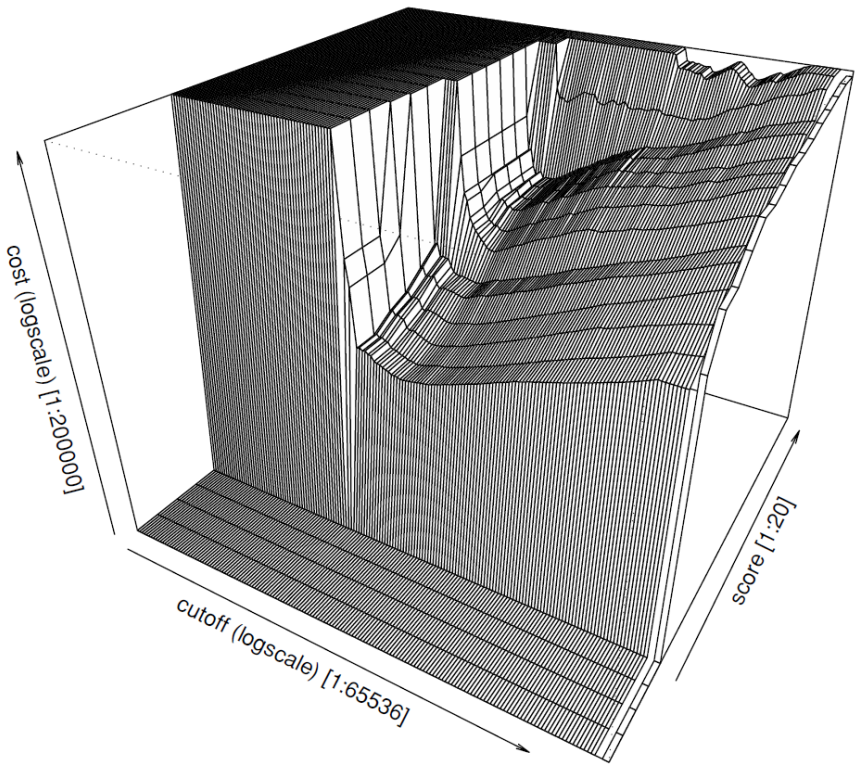
Modeling targets:

- Expected # backtracks, given a desired score $(P - N)$
- Expected score, given # backtracks

Top Down and Local (radial) Clause Search



Results

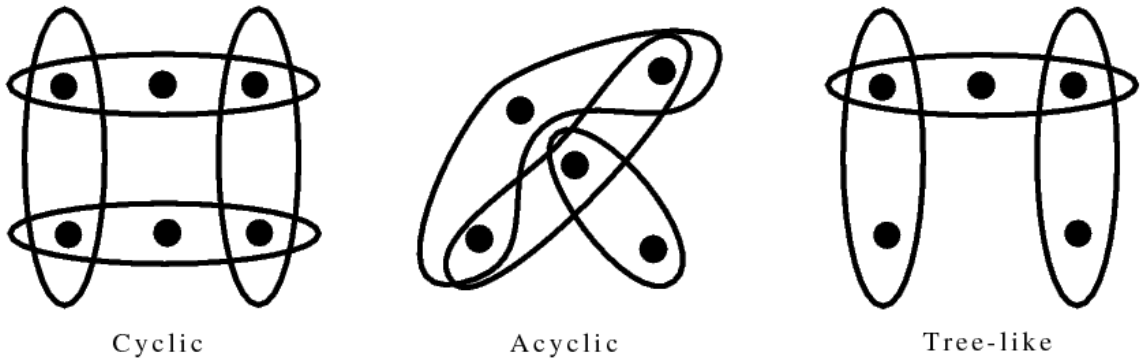


Same trends for all 5 methods * 3 domains

Exploiting Acyclicity in Clauses

Acyclicity

- a usual alleviating assumption in CSP and Database queries
- Not so common in ILP or FOL



variables $\approx$ vertices, atoms $\approx$ hyperedges

RelF

Relf algorithm [Kuzelka, Zelezny ICML 2009]
blockwise composition of clauses that are

- function-free, acyclic
- irreducible, irredundant, possibly very large

Reducibility example

$$grandson(x, y) \leftarrow son(x, z) \wedge son(x, w) \wedge son(z, y)$$

RelF

Relf algorithm [Kuzelka, Zelezny ICML 2009]
blockwise composition of clauses that are

- function-free, acyclic
- irreducible, irredundant, possibly very large

Reducibility example

$$grandson(x, y) \leftarrow son(x, z) \wedge son(z, y)$$

RelF

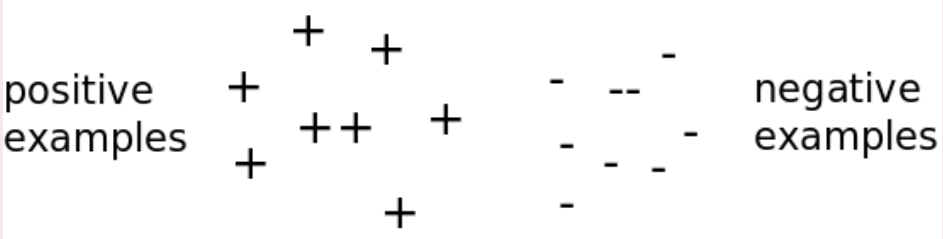
Relf algorithm [Kuzelka, Zelezny ICML 2009]
blockwise composition of clauses that are

- function-free, acyclic
- irreducible, irredundant, possibly very large

Reducibility example

$$grandson(x, y) \leftarrow son(x, z) \wedge son(z, y)$$

Redundancy example



RelF

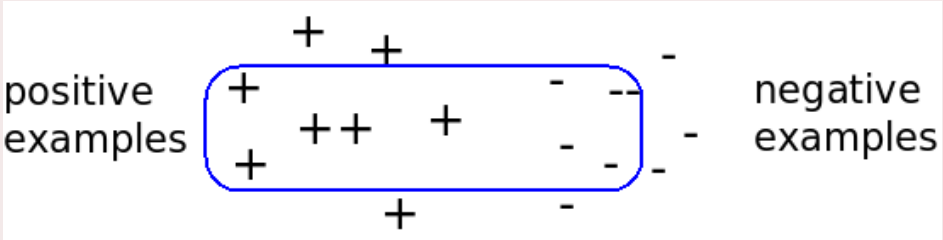
Relf algorithm [Kuzelka, Zelezny ICML 2009]
blockwise composition of clauses that are

- function-free, acyclic
- irreducible, irredundant, possibly very large

Reducibility example

$$grandson(x, y) \leftarrow son(x, z) \wedge son(z, y)$$

Redundancy example



RelF

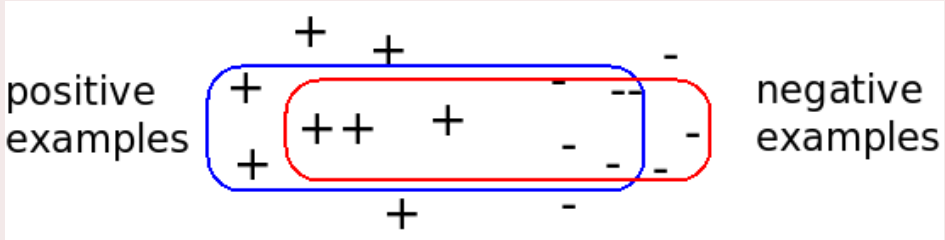
Relf algorithm [Kuzelka, Zelezny ICML 2009]
blockwise composition of clauses that are

- function-free, acyclic
- irreducible, irredundant, possibly very large

Reducibility example

$$grandson(x, y) \leftarrow son(x, z) \wedge son(z, y)$$

Redundancy example



RelF Performance

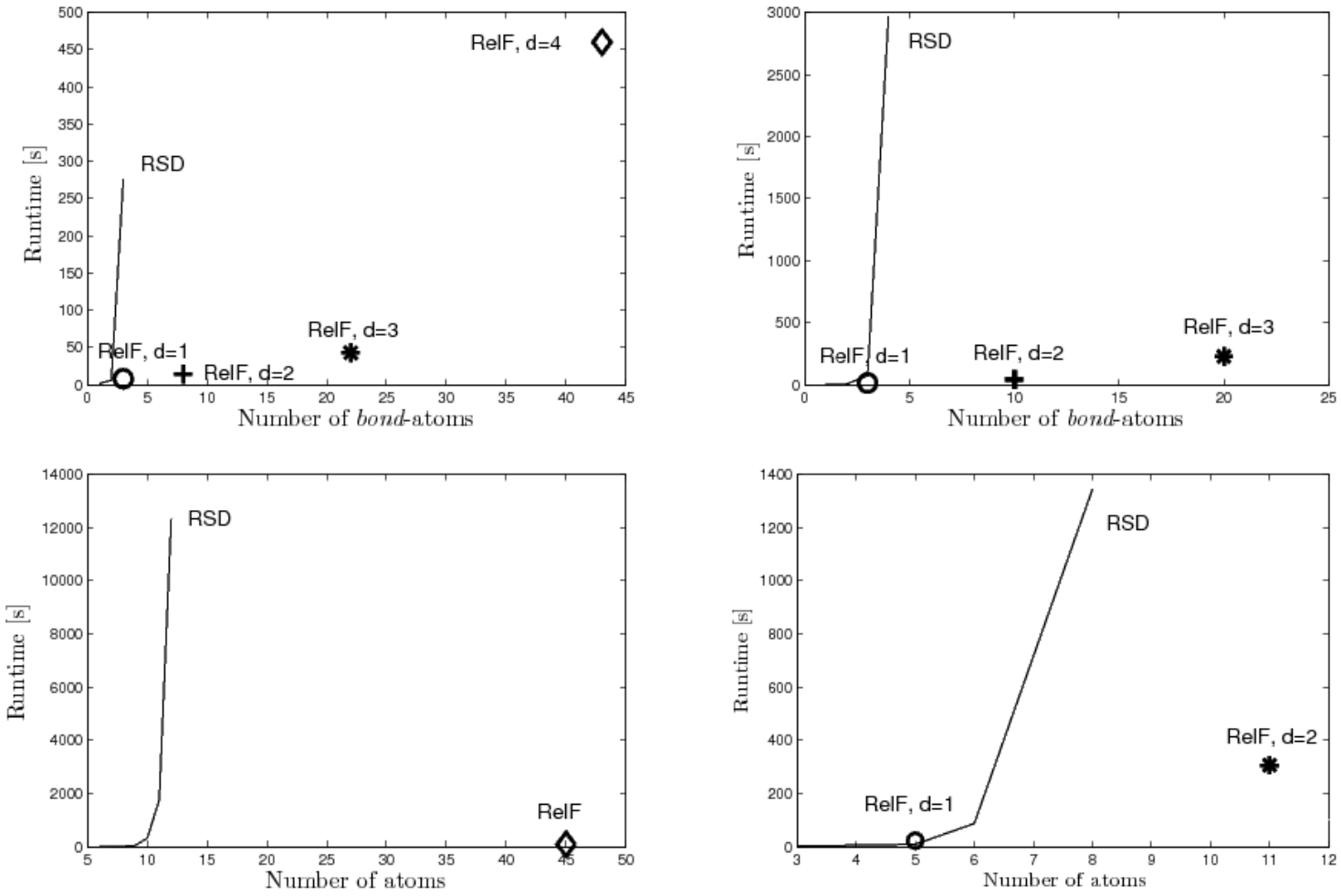


Fig. 2 Runtimes of RELF and RSD on Mutagenesis (top left), PTC (top right), CAD (bottom left) and Mesh

A CSP Model for Clause Search

- Previously: Maloberti's CSP model Django for subsumption. Fast.
- Why not a CSP model for clause search? But how?

A CSP Model for Clause Search

- Previously: Maloberti's CSP model Django for subsumption. Fast.
- Why not a CSP model for clause search? But how?
- Remind [Gottlob]:
  - finding a small consistent clause just as hard as finding a consistent $T\vartheta$
  - where T is a given template
- Our current (these days literally) work:
  - A CSP algorithm for finding ϑ
  - integrates subsumption testing ($T\vartheta\theta \subseteq e$)
  - [Barták, Železný, Kuželka - FLAIRS 2010], also submission to AAAI '10
- Open question: how to get template T?

Conclusions

Conclusions

- We are attacking two sources of ILP complexity
  - clause evaluation (subsumption check)
  - consistent clause search
- With all modesty, we are winning most of the battles

Conclusions

- We are attacking two sources of ILP complexity
  - clause evaluation (subsumption check)
  - consistent clause search
- With all modesty, we are winning most of the battles
- Our strategy is the "japanese miracle" one
  - exploit what other people have invented elsewhere
- in our case:
  - restarted randomized strategies (Gomes, Selman et al)
  - CSP heuristics (entire community)
  - acyclic bias (databases)

Acknowledgements

- Ashwin Srinivasan (Oxford U., IBM India)
- David Page (U of Wisconsin - Madison)
  - these two came up with the idea of restarted search in ILP
- Roman Barták (Charles Univ. Prague)
  - invented the CSP model for consistent clause search
- Czech Science Foundation Grant 201/08/0509 LeCoS: merging machine LEarning and COnstraint Satisfaction (2008-2010)
- Czech Science Foundation Grant 201/09/1665 ProLearn: Bridging the Gap between Systems Biology and Machine Learning (2009-2011)