From CATS to SAT:

Modeling Empirical Hardness to Understand and Solve Hard Computational Problems

Kevin Leyton-Brown

Computer Science Department

University of British Columbia

Intro

- From combinatorial auctions to supply chains and beyond, researchers in multiagent resource allocation frequently find themselves confronted with **hard computational problems**.
- This tutorial will focus on **empirical hardness models**, a machine learning methodology that can be used to predict how long an algorithm will take to solve a problem before it is run.

I. COMBINATORIAL AUCTIONS AND CATS

[Leyton-Brown, Pearson, Shoham, 2000]
[Leyton-Brown, 2003]

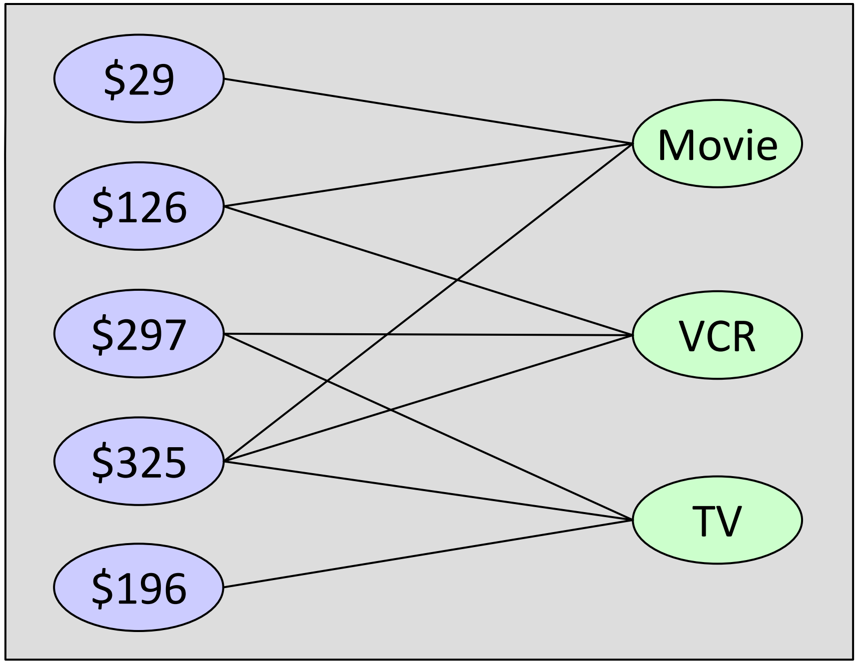
CATS

- My coauthors and I first developed this line of research in our work on the Combinatorial Auction Test Suite (CATS), when investigating whether "realistic" combinatorial auction problems were always computationally easier than the hardest artificial distributions.

- I’ll begin by describing CATS.

Combinatorial Auctions

- Auctions where bidders can request **bundles of goods**
  - Lately, a hot topic in CS
- Interesting because of **complementarity** and **substitutability**



Winner Determination Problem

- **Input**: n goods, m bids
  $< S_i, p_i >, S_i \subseteq \{1, \ldots, n\}$

- **Objective**: find revenue-maximizing non-conflicting allocation

$$
\begin{array}{lll}
\text{maximize:} & \sum_{i=1}^{m} x_i p_i & \\
\text{subject to:} & \sum_{i|g \in S_i} x_i \leq 1 & \forall g \\
 & x_i \in \{0, 1\} & \forall i
\end{array}
$$

What's known about WDP

Equivalent to **weighted set packing**, $\mathcal{NP}$-Complete

1. **Approximation**
   - best guarantee is within factor of $\sqrt{n}$
   - economic mechanisms can depend on optimal solution
2. **Polynomial special cases**
   - very few (ring; tree; totally unimodular matrices)
   - allowing unrestricted bidding is the whole point
3. **Complete heuristic search** (many examples exist; here are a few…)
   - CASS [Fujishima, Leyton-Brown, Shoham, 1999]
   - CABOB [Sandholm, 1999; Sandholm, Suri, Gilpen, Levine, 2001]
   - GL [Gonen & Lehmann, 2001]
   - CPLEX [ILOG Inc., 1987-2008]

Benchmark Data

- How should we judge a heuristic algorithm's **effectiveness** at solving the WDP?
- **Previous researchers** used:
  - **small-scale experiments** with human subjects, based on real economic problems
  - **artificial bid distributions** that can generate arbitrary amounts of data, but that lacked any economic motivation
- We proposed a **middle ground**: a test suite of artificial distributions that modeled real economic problems from the combinatorial auctions literature.

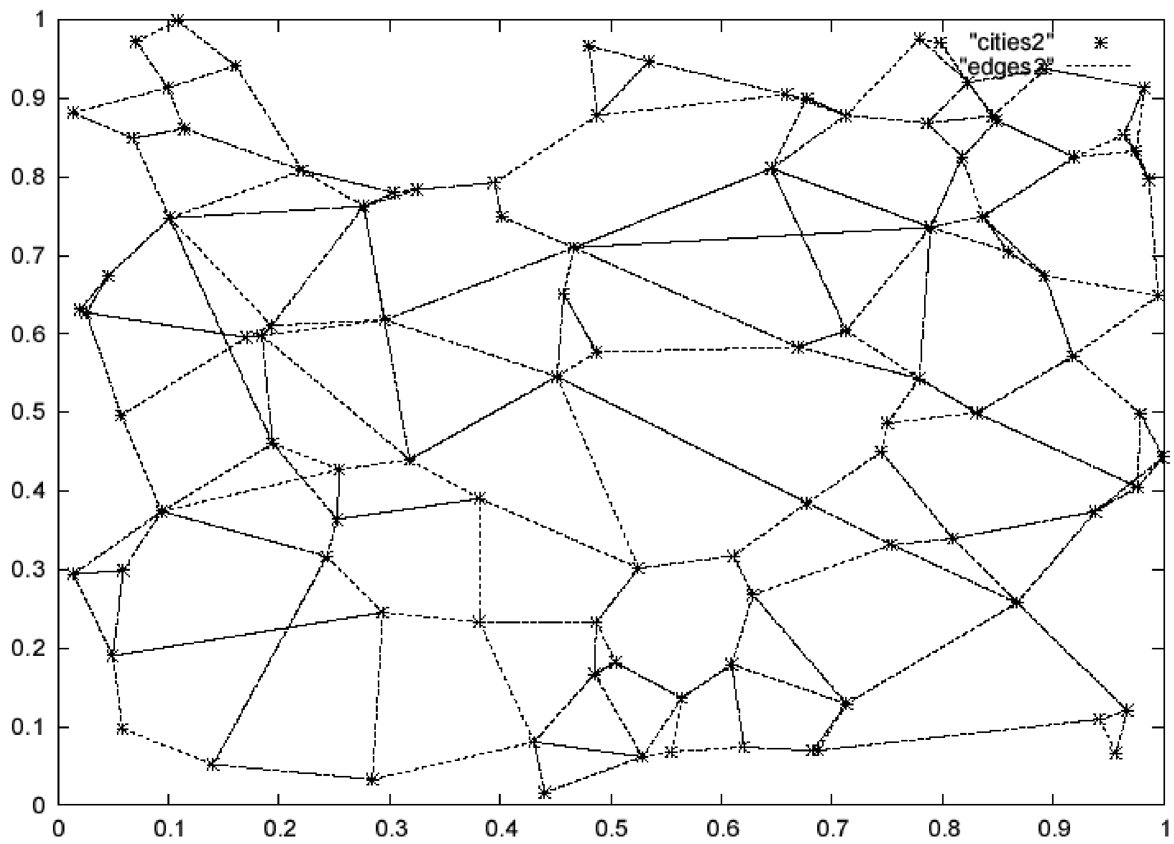
Combinatorial Auction Test Suite (CATS)

- Overall approach for **building a distribution**:
  - Identify a domain; basic bidder preferences
  - Derive an economic motivation for:
    - what goods bidders will request in bundle
    - how bidders will value goods in a bundle
    - what bundles form sets of substitutable bids
  - **Key question**: from what does complementarity arise?

- The **CATS distributions** [Leyton-Brown, Pearson, Shoham, 2000]:
  1. Paths in space
  2. Proximity in space
  3. Arbitrary relationships
  4. Temporal Separation (matching)
  5. Temporal Adjacency (scheduling)

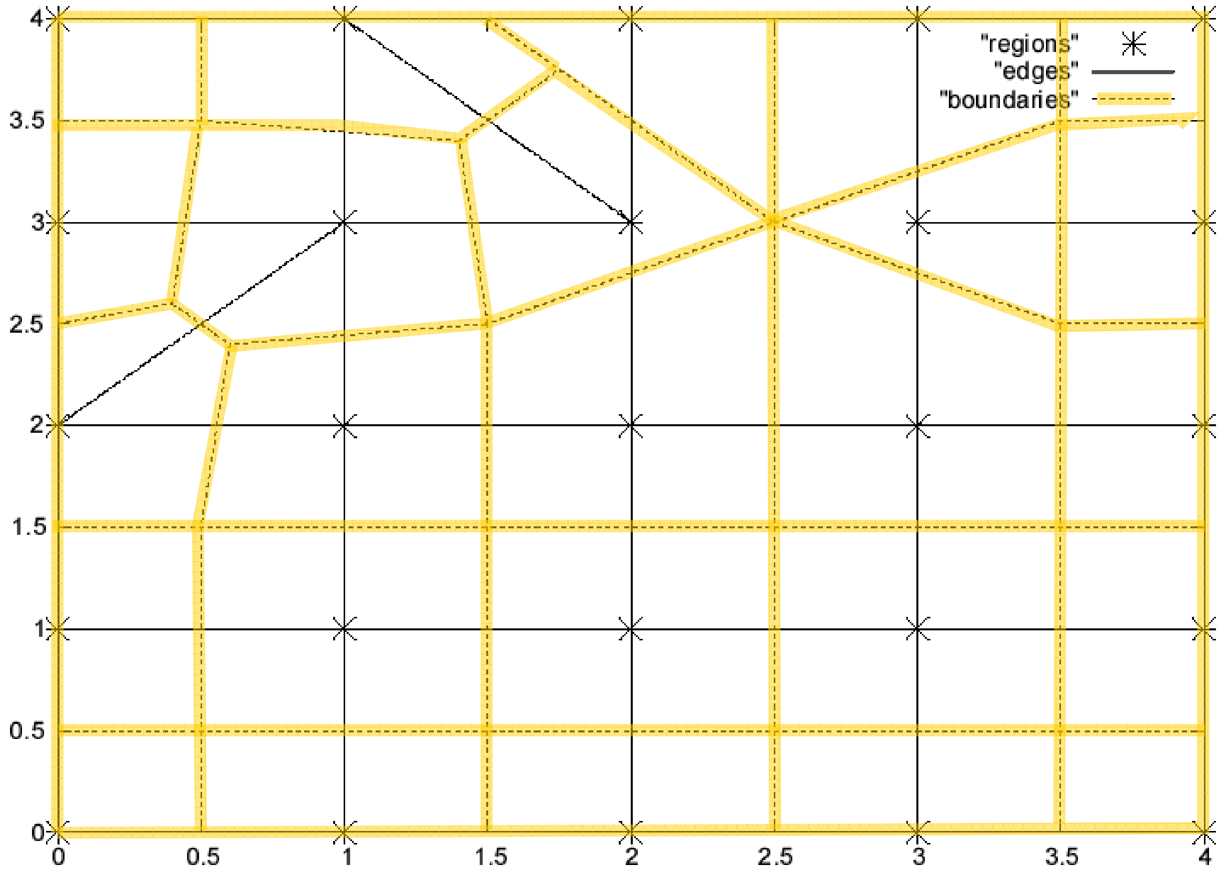
Example Distribution: Paths in Space

- Model bidders who want to buy a **route in a network**
- Generate a **planar graph**; bid on a set of **short paths**



Example Distribution: Regions in Space

- Generate a **graph based on a grid**
- Bidders request sets of **adjacent vertices**

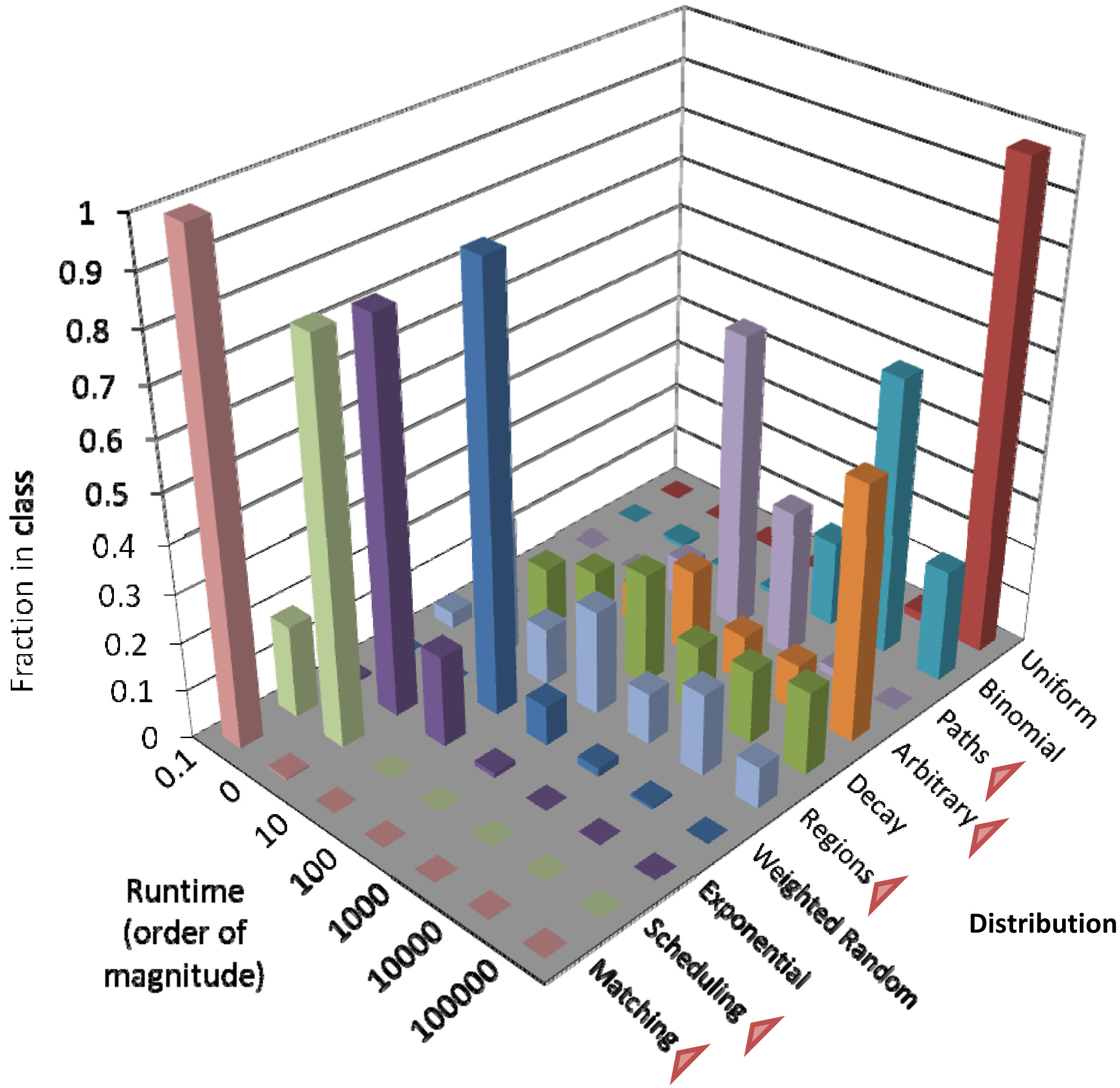


Other CATS Distributions

- **Arbitrary Relationships**:
  - a generalization of Regions that begins with a complete graph
- **Temporal Matching**:
  - a model of aircraft take-off / landing slot auctions
- **Temporal Scheduling**:
  - a model of job-shop scheduling
- **Legacy Distributions**:
  - nine of the artificial distributions that were widely used before

How Hard is CATS?

(CPLEX 7.1, 550 MHz Xeon; 256 goods, 1000 bids)



Questions About CATS

- CATS has become widely used as a way of **evaluating WDP algorithms**
  - also used for a purpose we didn't expect: modeling agent preferences for uses other than evaluating WDP algorithms
- Some researchers found that their algorithms were **much faster on CATS** than on certain legacy distributions
  - did this mean that **real CA problems are easier** than the hardest artificial problems?
  - did this just mean that **the CATS distributions were easy**?
  - did this mean that we had **chosen the wrong parameters** for some of the CATS distributions?
- Another phenomenon: even top algorithms like CPLEX are **blindingly fast** on some instances; **incredibly slow** on others.

II. EMPIRICAL HARDNESS MODELS FOR COMBINATORIAL AUCTIONS

[Leyton-Brown, Nudelman, Shoham, 2002]

[Leyton-Brown, Nudelman, Andrew, McFadden, Shoham, 2003]

[Leyton-Brown, Nudelman, Andrew, McFadden, Shoham, 2003]

[Leyton-Brown, Nudelman, Shoham, 2008]

Empirical Hardness Models

- To see if we'd made CATS too easy, we investigated **tuning CATS' generators** to create harder instances.
- Along the way, we developed a host of other methods that I will survey today:
  - accurately **predicting an algorithm's runtime** on an unseen instance
  - determining **which instance properties** most affect an algorithm's performance
  - building **algorithm portfolios** that can dramatically outperform their constituent algorithms

Empirical Hardness Methodology

1. Select **algorithm**
2. Select set of **distributions**
3. Select **features**
4. Generate **instances**
5. **Compute** running time, features
6. **Learn** running time model

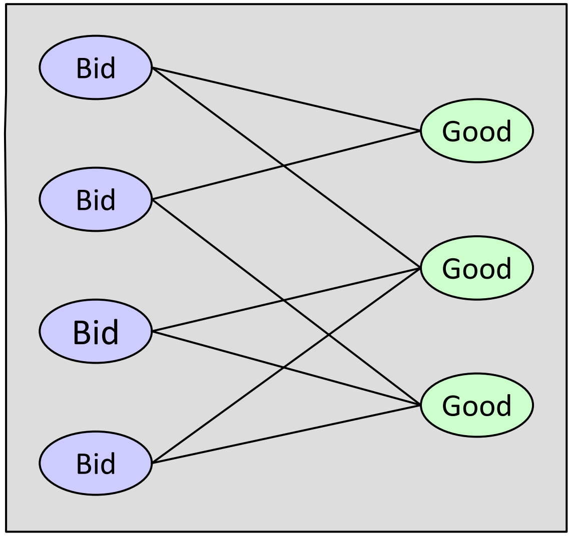
Features

1. **Linear Programming**
   – L_1, L_2, L_∞ norms of integer slack vector

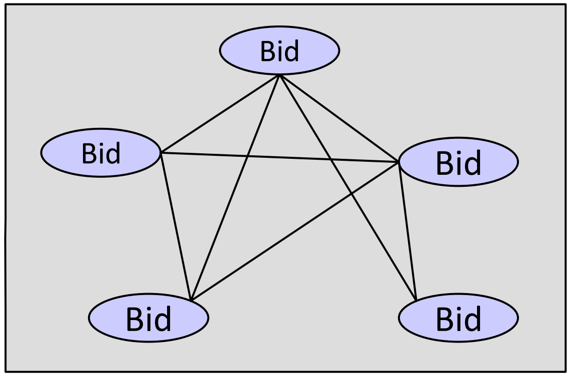
$$\text{maximize:} \quad \sum_{i=1}^{m} x_i p_i$$
$$\text{subject to:} \quad \sum_{i|g\in S_i} x_i \le 1 \quad \forall g$$
$$0 \le x_i \le 1 \quad \forall i$$

2. **Price**
   – stdev(prices)
   – stdev(avg price / num goods)
   – stdev(average price / sqrt(num goods))

3. **Bid-Good graph**
   – node degree stats (max, min, avg, stdev)



4. **Bid graph**
   – node degree stats
   – edge density
   – clustering coefficient (CC), stdev
   – avg min path length (AMPL)
   – ratio of CC to AMPL
   – eccentricity stats (max, min, avg, stdev)



Building Empirical Hardness Models

- A **set of instances** D
- For each instance $i \in D$, a vector $\boldsymbol{x}_i$ of **feature values**
- For each instance $i \in D$, a **runtime observation** y_i

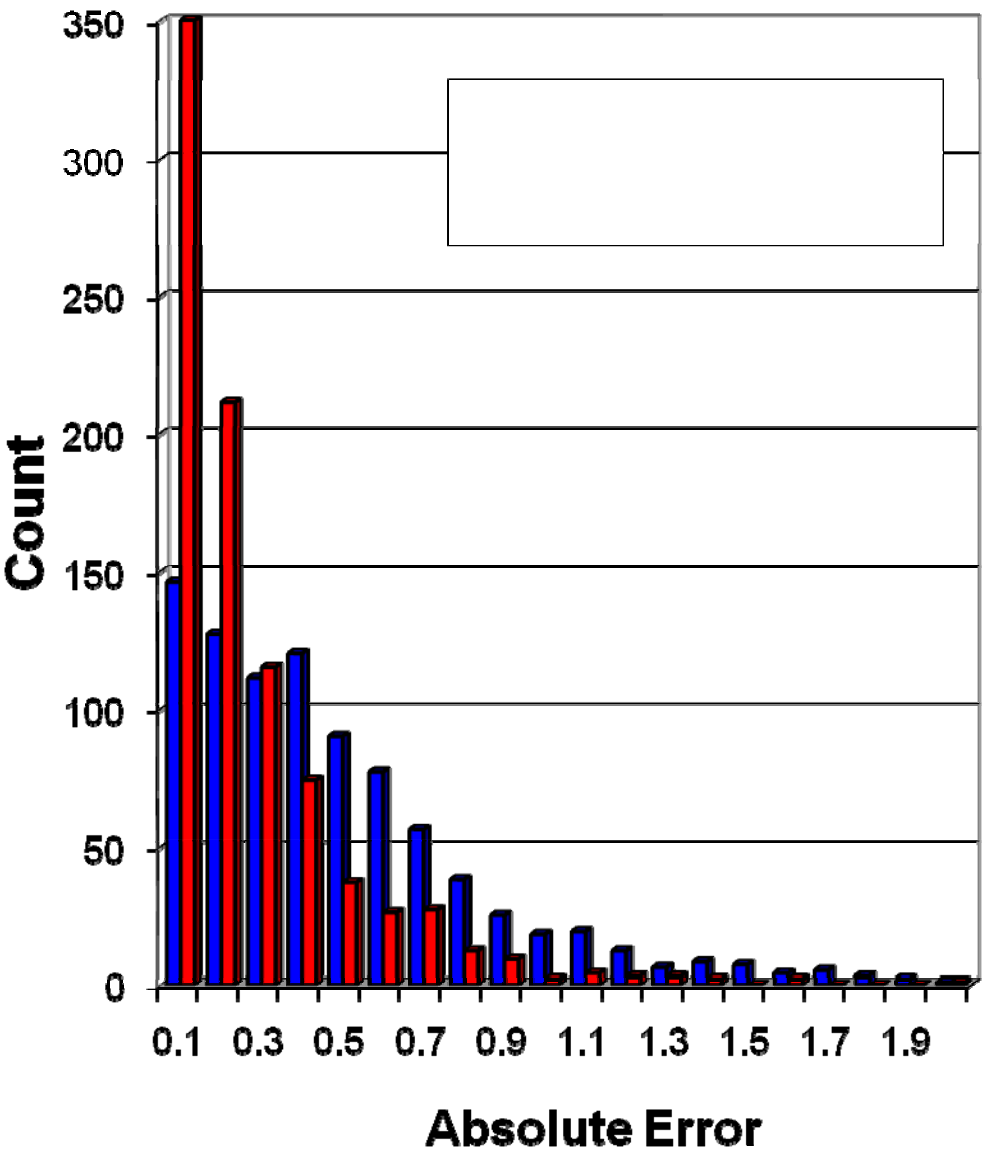
- We want a mapping $f(x) \mapsto y$ that **accurately predicts** y_i given $\boldsymbol{x}_i$
  - This is a **regression** problem
  - We've tried various methods:
    - Gaussian process regression
    - boosted regression trees
    - lasso regression
    - ...
  - Overall, we've achieved high accuracy combined with tractable computation by using **basis function ridge regression**

Building a Regression Model

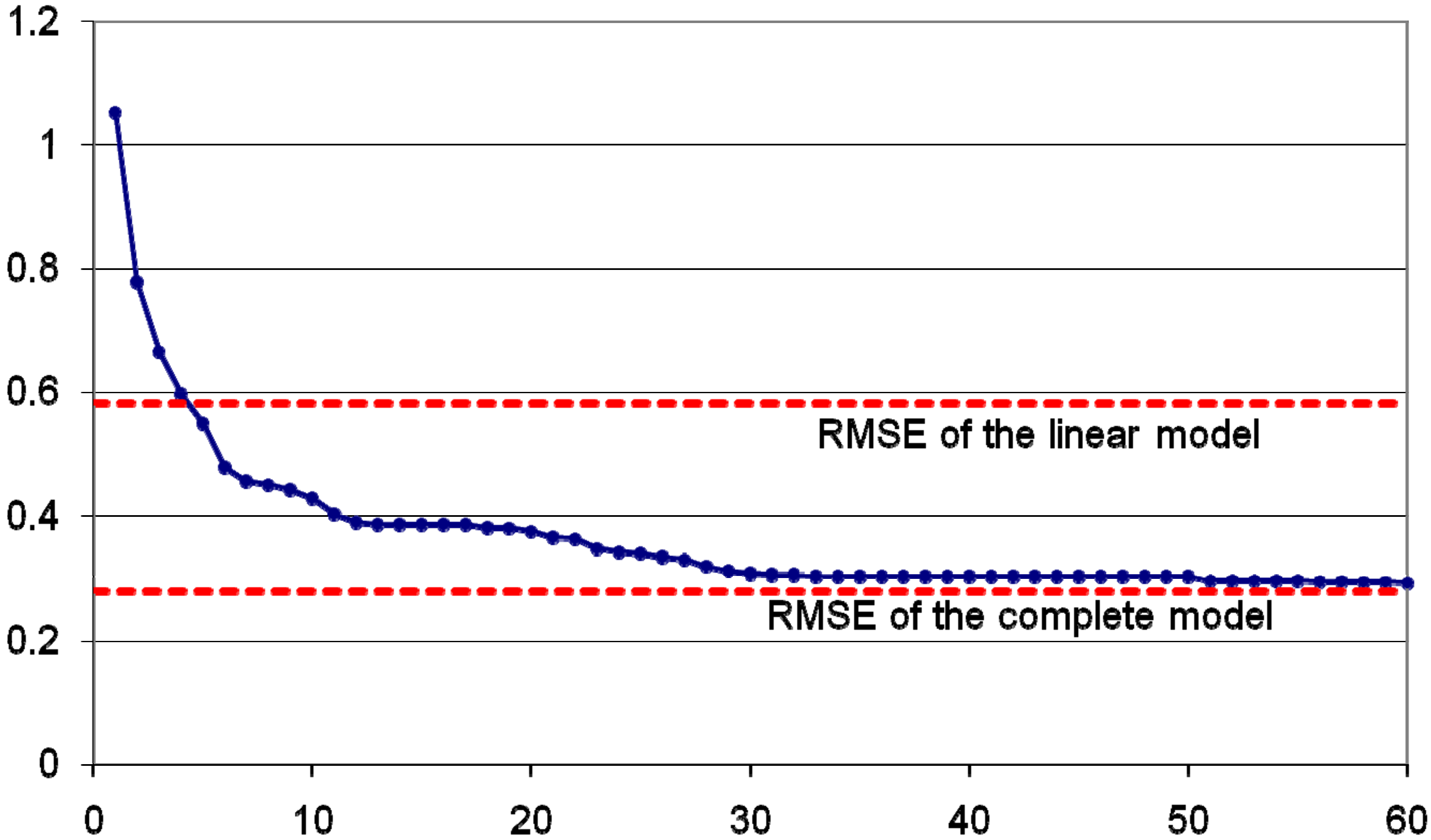
1. **log transform runtime**: set $y = \log_{10}(y)$
2. **forward selection**: discard unnecessary features from $\boldsymbol{x}$
3. **add new features** by performing a basis function expansion of the existing features
   - $\phi_i = [\phi_1(\boldsymbol{x}_1), \ldots, \phi_k(\boldsymbol{x}_k)]$
4. run **another pass of forward selection** on $\Phi = [\phi_1, \ldots, \phi_k]$
5. use **ridge regression** to learn a linear function of the basis function expansion of the features
   - let δ be a small constant (e.g., 10^{-3})
   - $w = (\delta I + \Phi^\top \Phi)^{-1} \Phi^\top y$
   - to predict $\log_{10}$(runtime), evaluate $w^\top \phi(\boldsymbol{x}_i)$

Learning

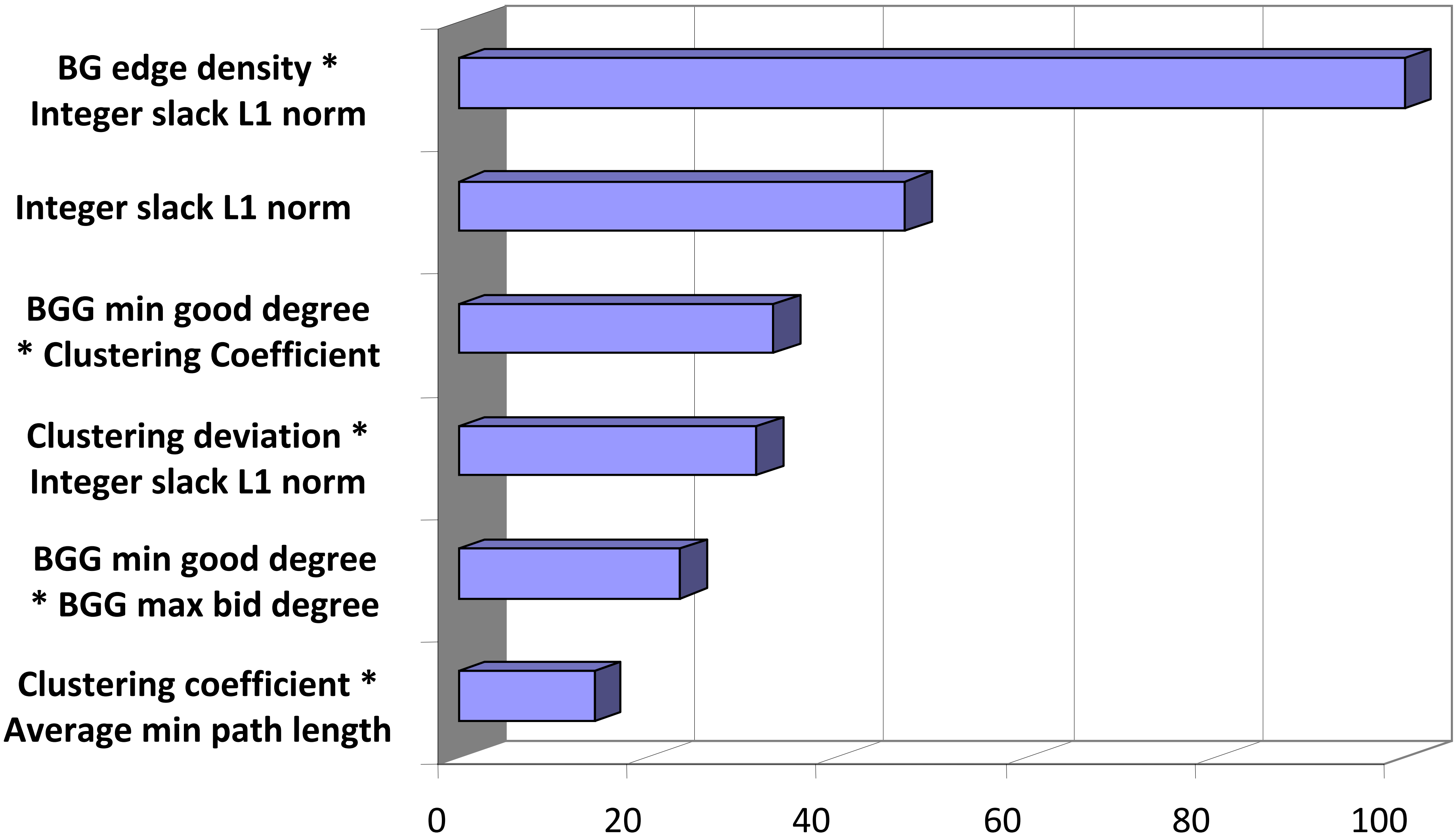
- Linear ridge regression
  - ignores interactions between variables
- Consider **2nd degree polynomials**
  - basis functions: pairwise products of original features
  - total of 325
- We tried various other non-linear approaches; **none worked better**.



Understanding Models: RMSE vs. Subset Size



Cost of Omission (subset size 6)



Boosting as a Metaphor for Algorithm Design

[Leyton-Brown, Nudelman, Andrew, McFadden, Shoham, 2003]

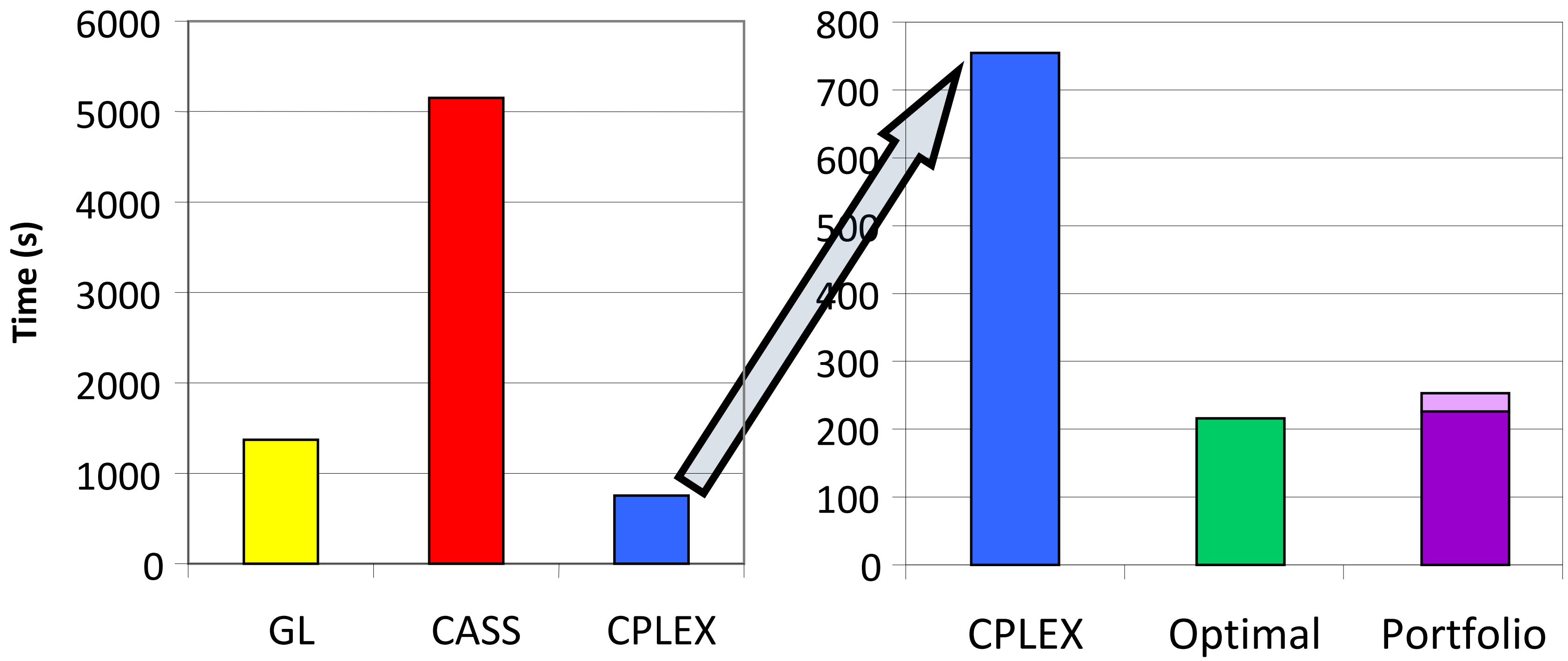
Boosting (machine learning technique):

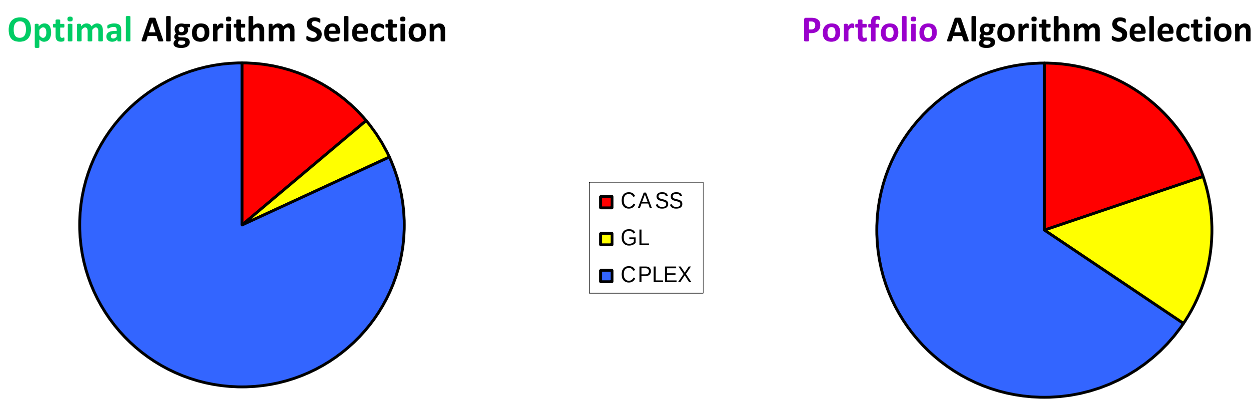
1. Combine uncorrelated weak classifiers into aggregate
2. Train new classifiers on instances that are hard for the aggregate

Algorithm Design with Hardness Models:

1. Hardness models can be used to **select an algorithm** to run on a per-instance basis
2. Use portfolio hardness model as a PDF, to **induce a new test distribution** for design of new algorithms

Portfolio Results



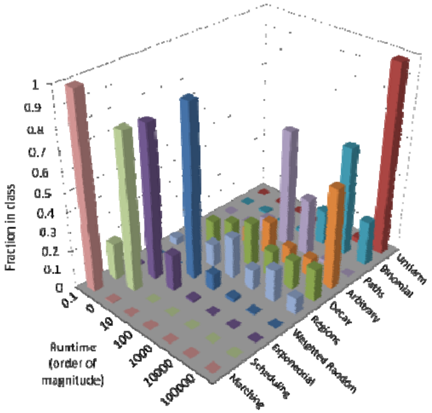


Distribution Induction

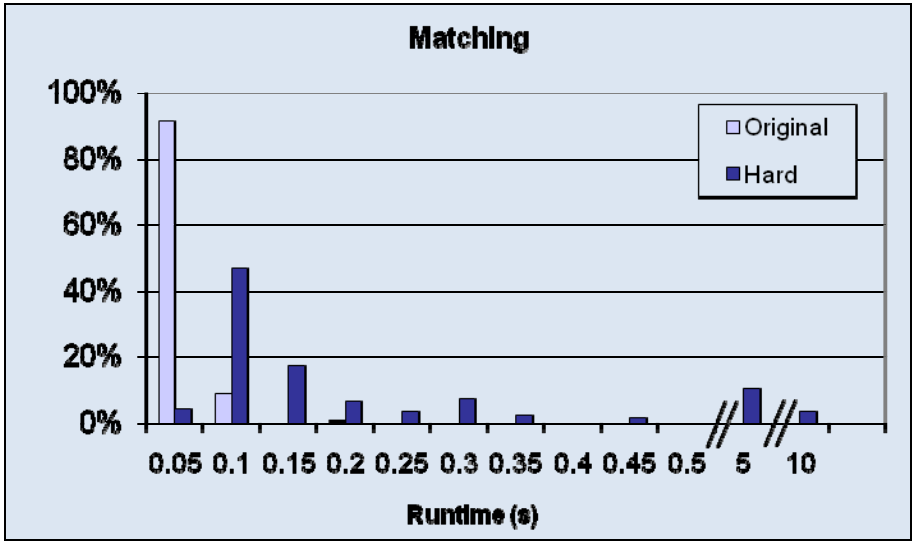
- We want our test distribution to generate problems **in proportion to the time our portfolio spends** on them
  - D: original distribution of instances
  - H_f: model of portfolio runtime (h_f: normalized)
- Goal: **generate instances** from $D \leq h_f$
  - D is a distribution over the parameters of an instance generator
  - h_f depends on features of generated instance
- **Rejection sampling**
  1. Create model of hardness H_p using parameters of the instance generator as features; normalize it to create a PDF h_p
  2. Generate an instance from $D \leq h_p$
  3. Keep the sample with probability proportional to $\frac{H_f(s)}{h_p(s)}$

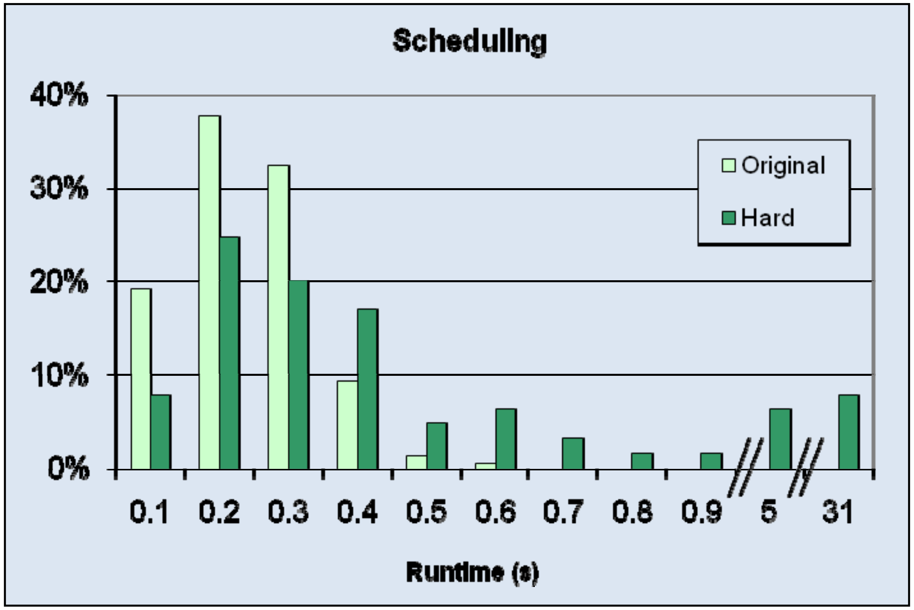
Distribution Induction

- Wide spread of runtimes in D, high accuracy of H_f
  - induction is easy



- Demonstrate our techniques on more challenging settings with small variance
  - matching, scheduling





III. EMPIRICAL HARDNESS MODELS FOR SAT

[Nudelman, Leyton-Brown, Devkar, Hoos, Shoham, 2004]
[Hutter, Hamadi, Hoos, Leyton-Brown, 2006]
[Xu, Hoos, Leyton-Brown, 2007]

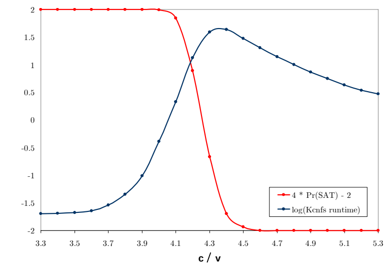
some slides based on originals by Eugene Nudelman

Empirical Hardness Models for SAT

- After establishing to ourselves that empirical hardness models are a useful way to tackle combinatorial auction problems, we sought to demonstrate their effectiveness on a **more widely-studied NP-complete problem**
- Thus, **we turned to SAT**
  - also interesting: it is a decision, not optimization problem
  - (especially) uniform-random 3-SAT has been widely studied
- After discussing our models, I'll describe some of the **new techniques** we developed for SAT:
  - the **direct prediction of satisfiability status**
  - the construction of **hierarchical models**
  - dealing with **censored data**

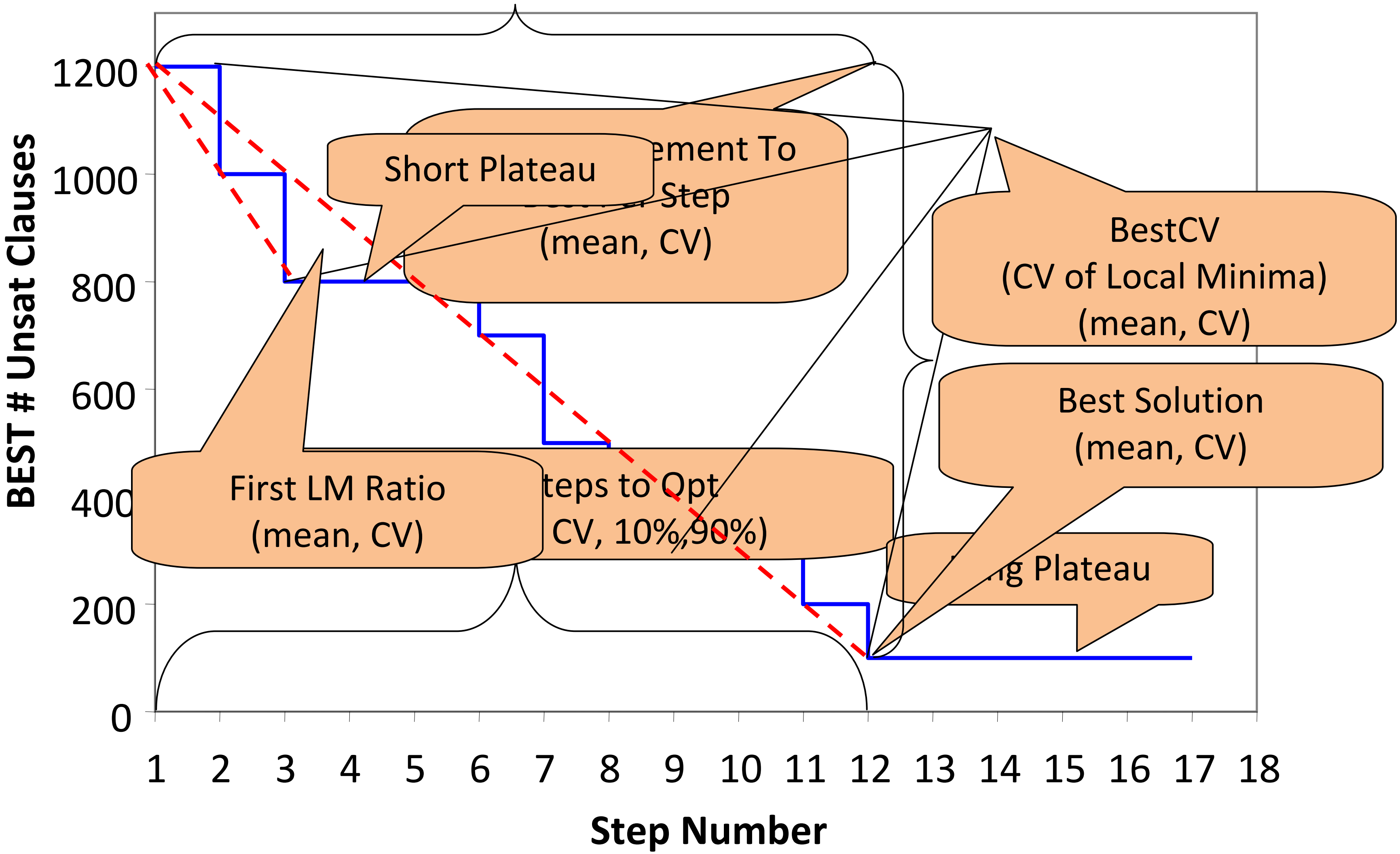
Previously…

- Easy-hard-less hard transitions discovered in the behaviour of DPLL-type solvers [Selman, Mitchell, Levesque]
  - Strongly correlated with phase transition in solvability
  - Spawned a new enthusiasm for using empirical methods to study algorithm performance

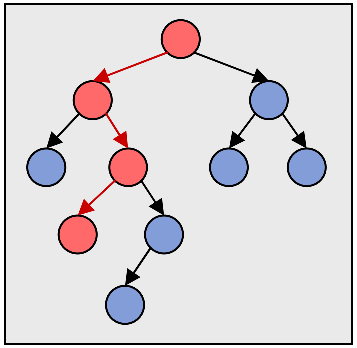


- Follow-up has included study of:
  - Islands of tractability [Kolaitis et. al.]
  - SLS search space topologies [Frank et.al., Hoos]
  - Backbones [Monasson et.al., Walsh and Slaney]
  - Backdoors [Williams et. al.]
  - Random restarts [Gomes et. al.]
  - Restart policies [Horvitz et.al, Ruan et.al.]
  - …

Features: Local Search Probing



Features: DPLL, LP



- **DPLL** search space size estimate
  - **Random probing** with unit propagation
  - Compute mean depth till contradiction
  - Estimate log(#nodes)

- Cumulative number of **unit propagations** at different depths (DPLL with Satz heuristic)

- **LP relaxation**
  - Objective value
  - stats of integer slacks
  - #vars set to an integer

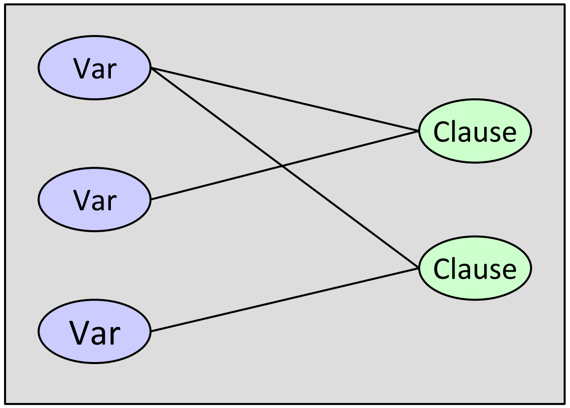
$$
\begin{array}{lll}
\text{maximize:} & \displaystyle\sum_{k\in C}\left(\sum_{i\in L,\, i\in k} v_i + \sum_{j\in \overline{L},\, i\in k} (1-v_j)\right) & \\
\text{subject to:} & \displaystyle\sum_{i\in k,\, i\in L} v_i + \sum_{j\in k,\, j\in \overline{L}} (1-v_j) \geq 1 & \forall k\in C \\
 & v_i \in \{0,1\} & \forall i
\end{array}
$$

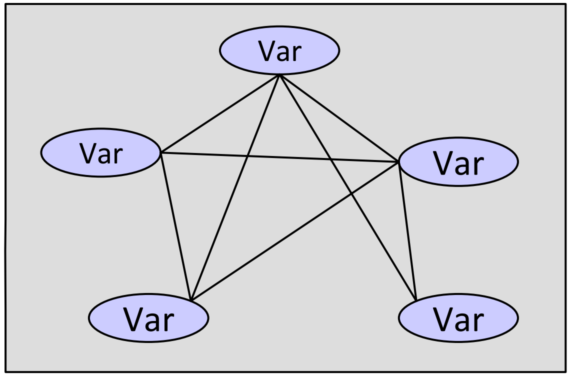
Other Features

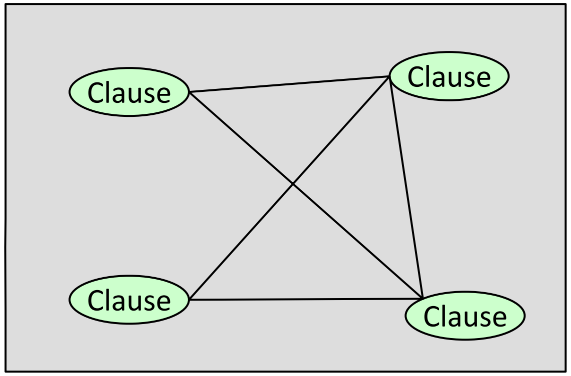
- **Problem Size**:
  - v (#vars)
  - c (#clauses)
  - Powers of c/v, v/c, ♣c/v - 4.26♣

used for normalizing many other features

- **Graphs**:
  - **Variable-Clause** (VCG, bipartite)
  - **Variable** (VG, edge whenever two variables occur in the same clause)
  - **Clause** (CG, edge iff two clauses share a variable with opposite sign)
- **Balance**
  - #pos vs. #neg literals
  - unary, binary, ternary clauses
- Proximity to **Horn formula**



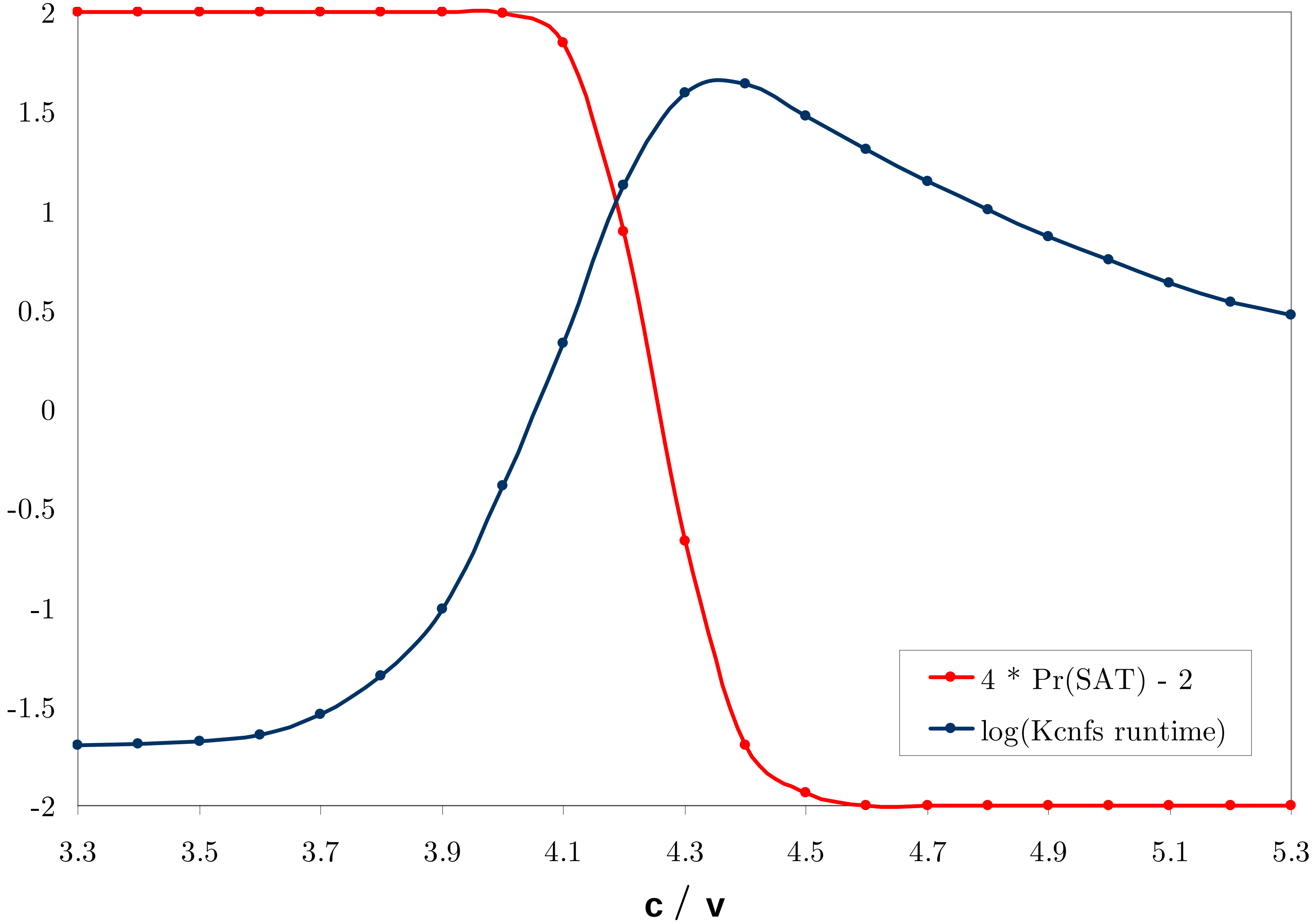




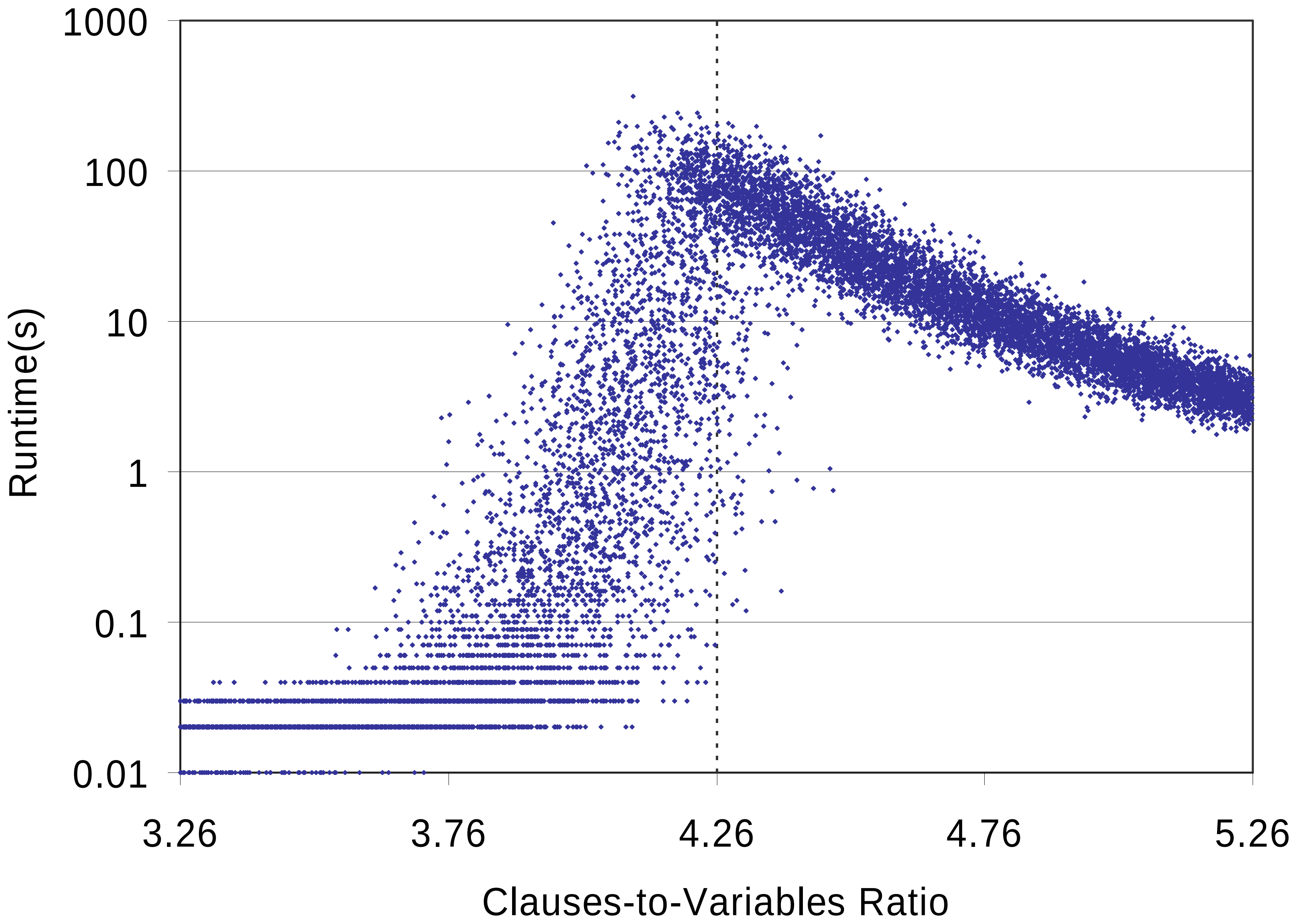
Experiments on Uniform-Random 3-SAT

- Uniform random 3-SAT, 400 vars
- **Datasets** (20000 instances each)
  - **Variable-ratio** dataset (1 CPU-month)
    - c/v uniform in [3.26, 5.26] ($\therefore c \in [1304,2104]$)
  - **Fixed-ratio** dataset (4 CPU-months)
    - c/v=4.26 ($\therefore v$=400, c=1704)
- **Solvers**
  - Kcnfs [Dubois and Dequen]
  - OKsolver [Kullmann]
  - Satz [Chu Min Li]
- **Quadratic basis function ridge regression**
- Training : test : validation split was 70 : 15 : 15

Kcnfs Data

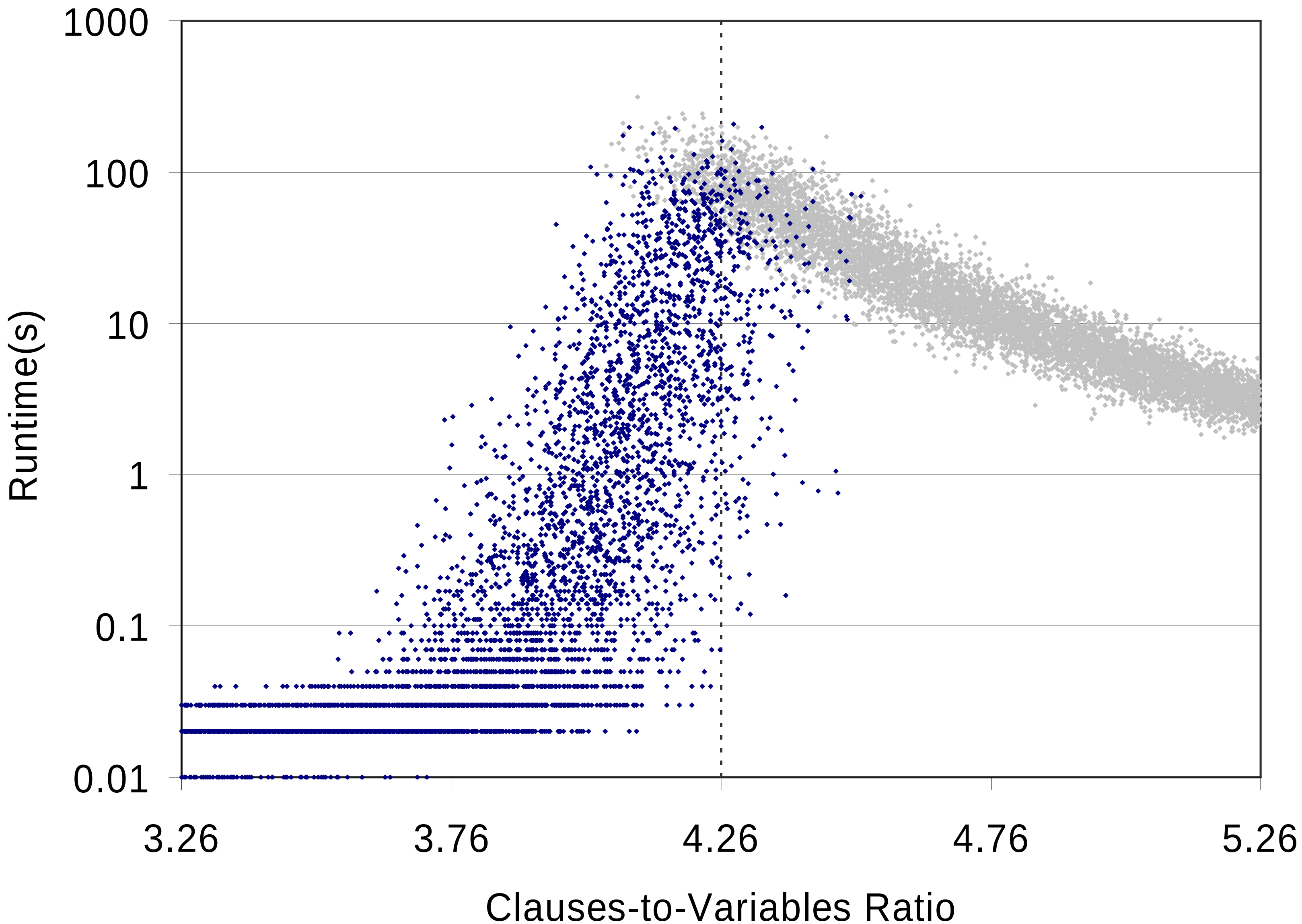

2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
3.3
3.5
3.7
3.9
4.1
4.3
4.5
4.7
4.9
5.1
5.3
c / v
4 * Pr(SAT) - 2
log(Kcnfs runtime)

Kcnfs Data

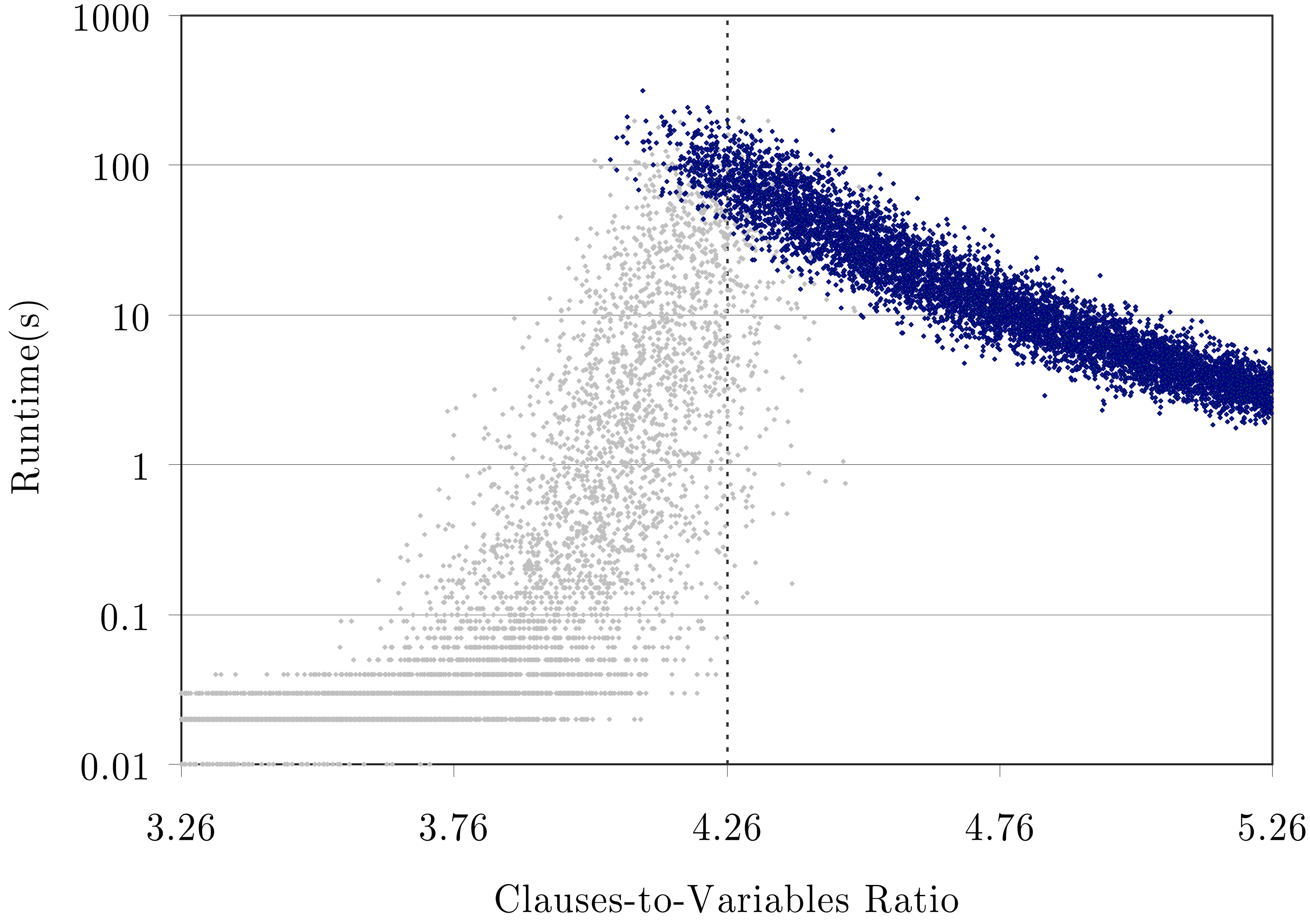


1000
100
10
1
0.1
0.01
Runtime(s)
3.26
3.76
4.26
4.76
5.26
Clauses-to-Variables Ratio

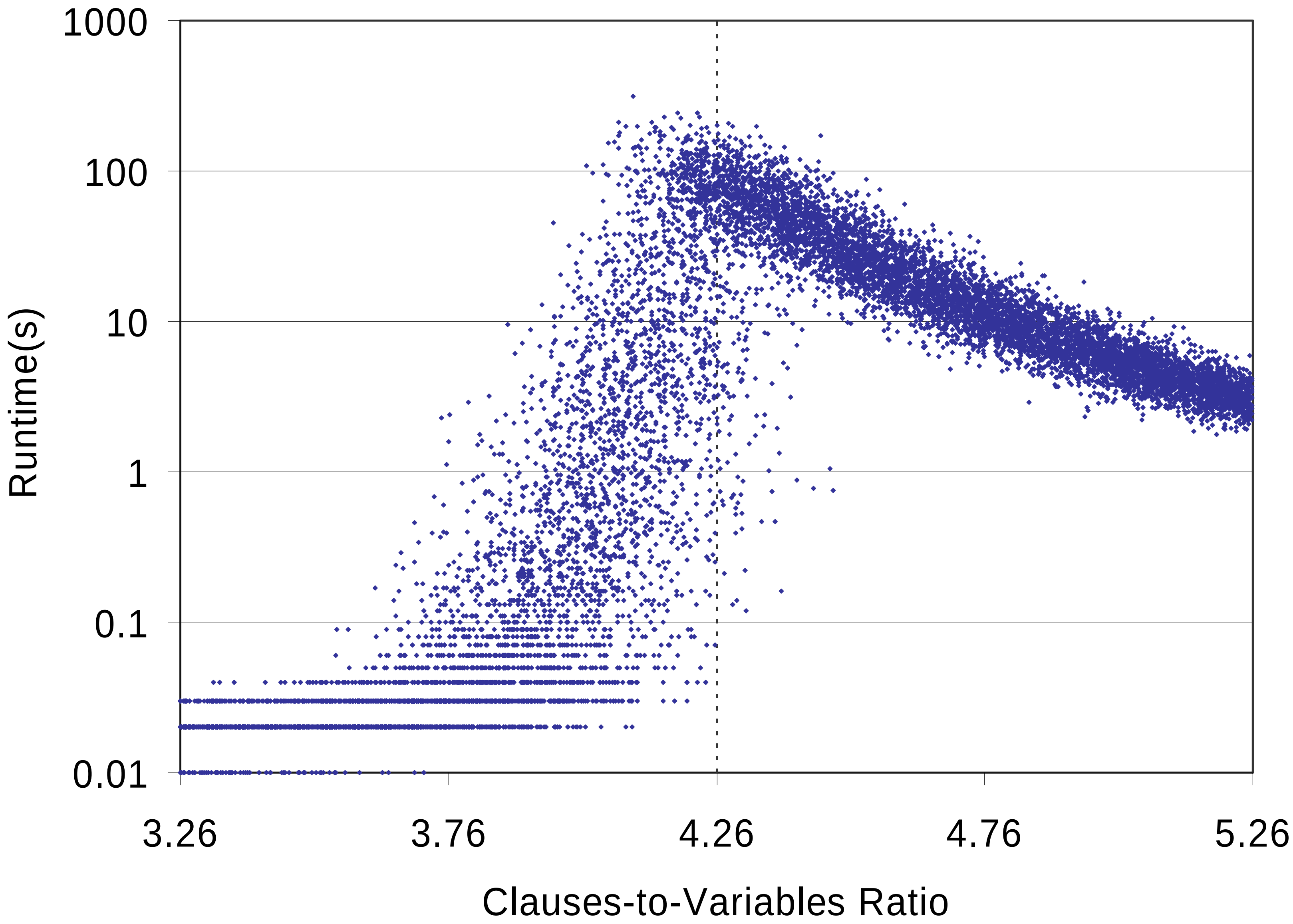
Kcnfs Data


1000
100
10
1
0.1
0.01
Runtime(s)
3.26
3.76
4.26
4.76
5.26
Clauses-to-Variables Ratio

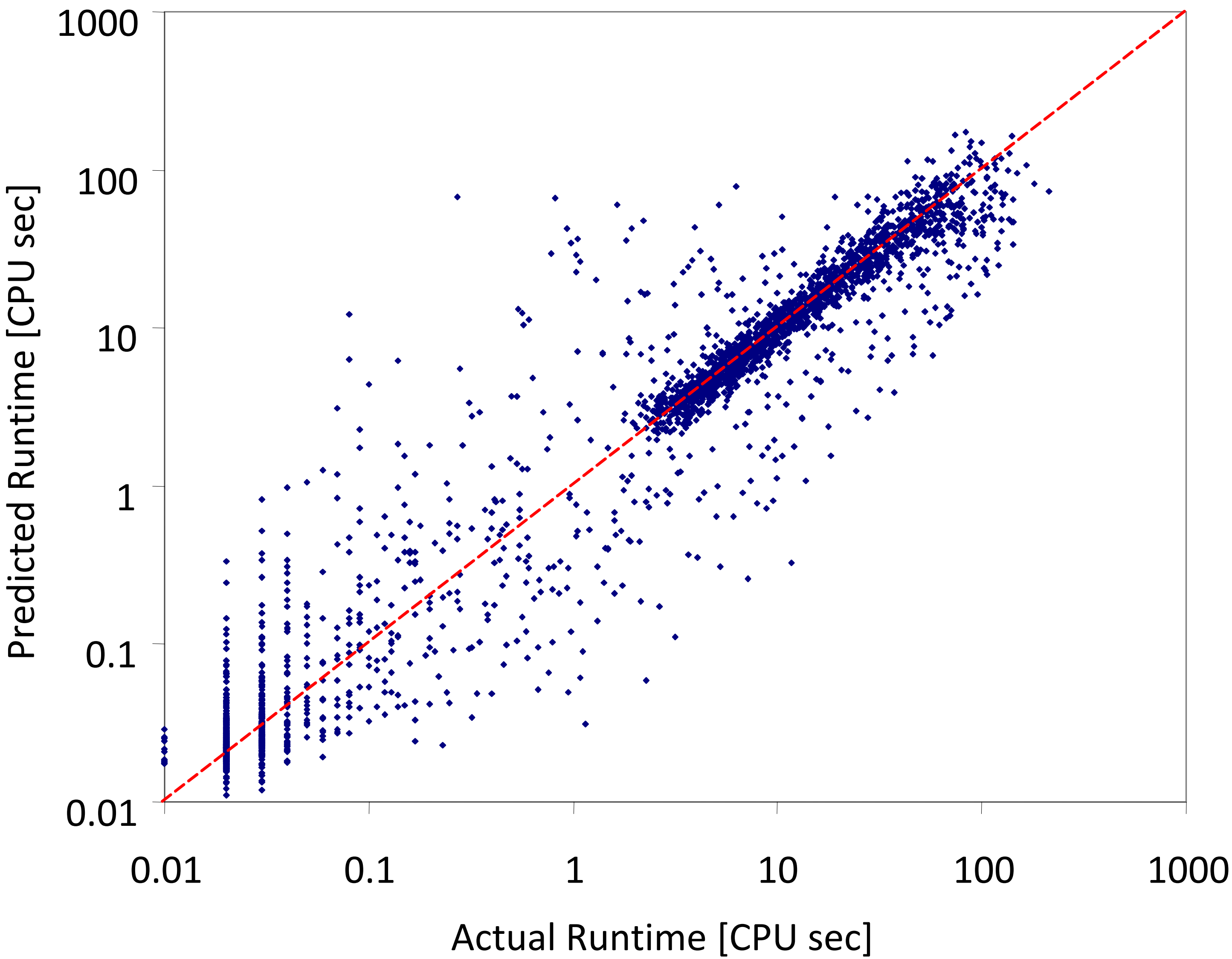
Kcnfs Data


1000
100
10
1
0.1
0.01
Runtime(s)
3.26
3.76
4.26
4.76
5.26
Clauses-to-Variables Ratio

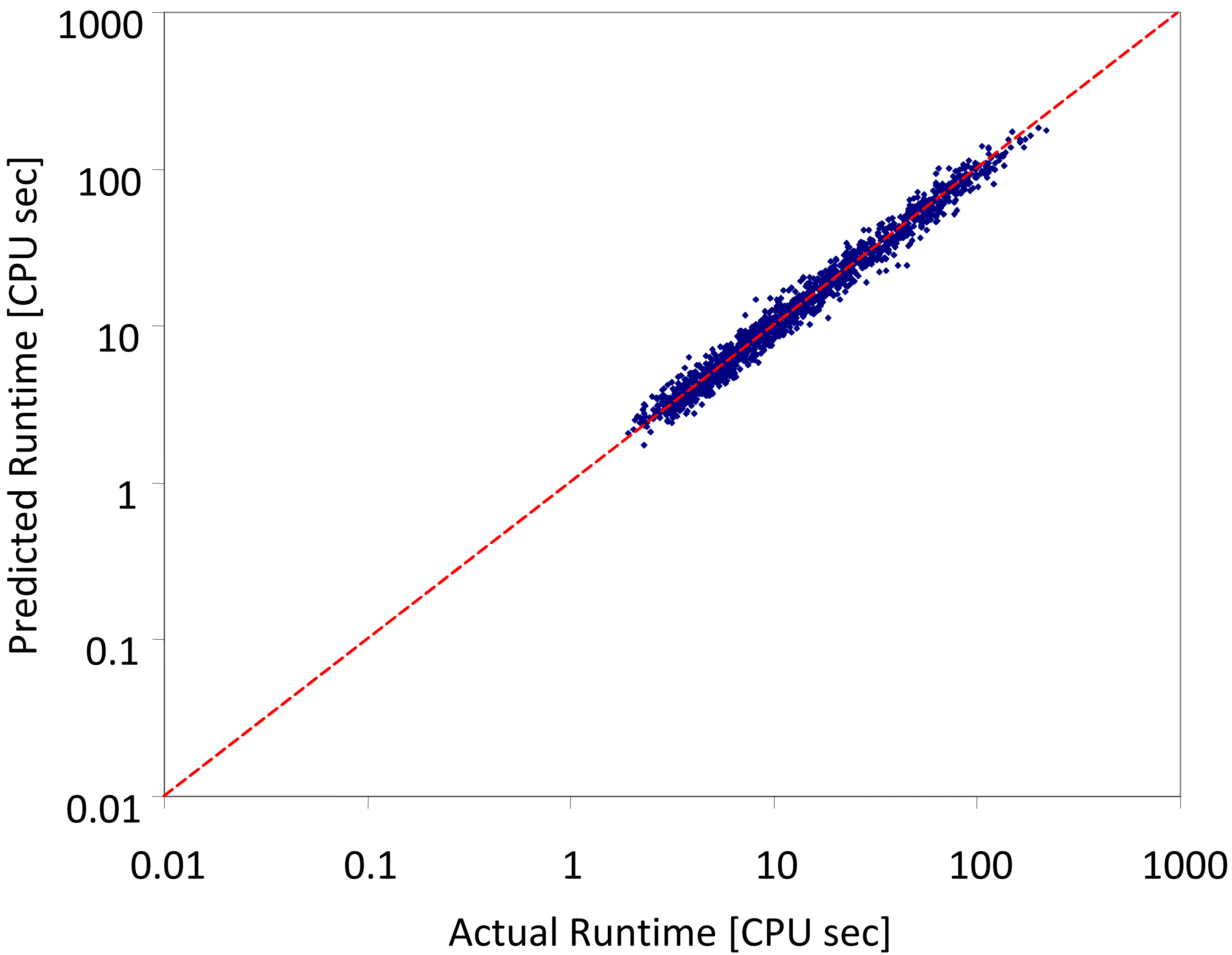
Kcnfs Data


1000
100
10
1
0.1
0.01
Runtime(s)
3.26
3.76
4.26
4.76
5.26
Clauses-to-Variables Ratio

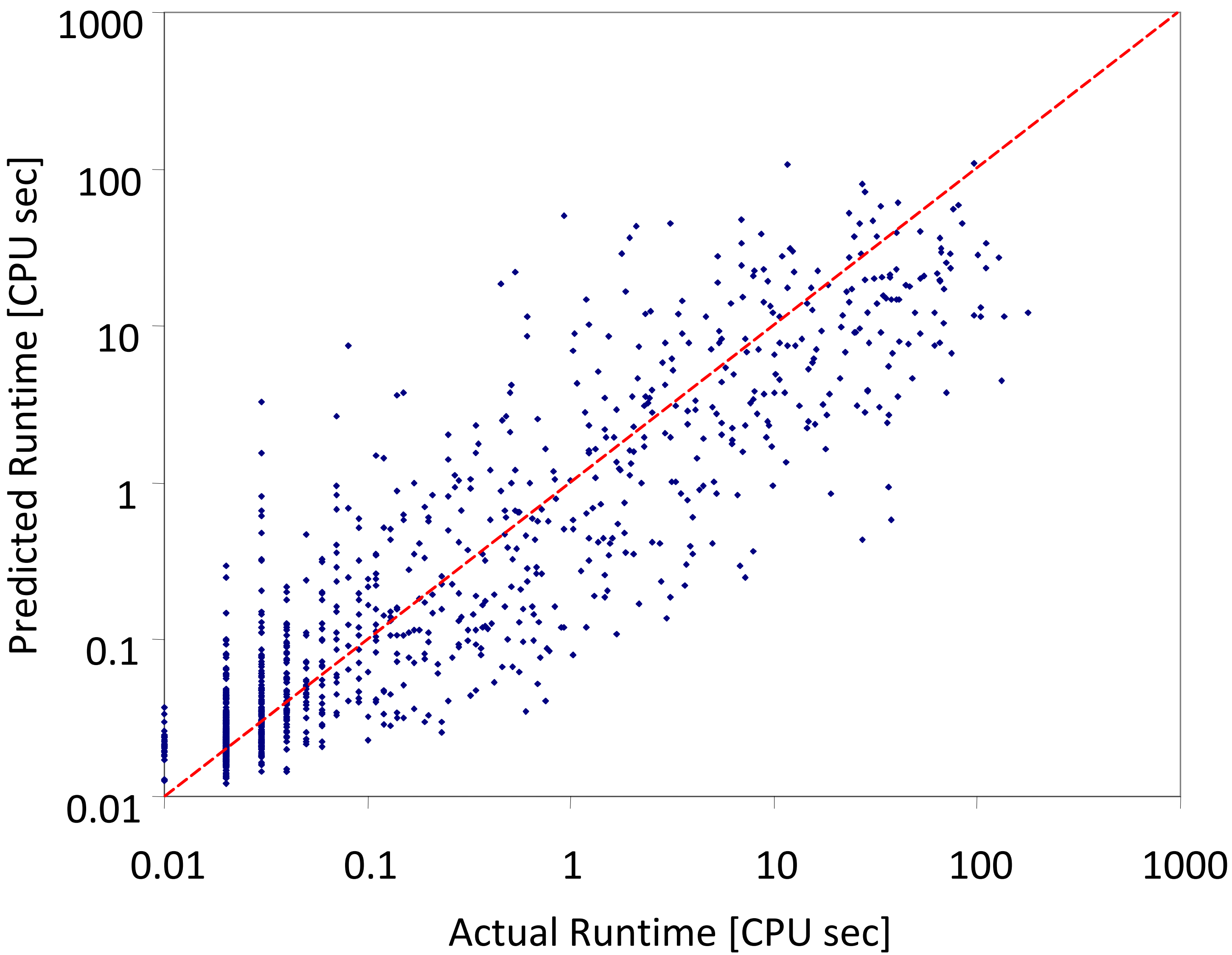
Variable Ratio Prediction (Kcnfs)



Variable Ratio - UNSAT



Variable Ratio - SAT



Feature Importance – Variable Ratio

- **Subset selection** was used to identify features **sufficient** for achieving good performance
- As before, other (correlated) subsets could potentially achieve similar performance

Variable	Cost of Omission
$\clubsuit c/v$ - $4.26\clubsuit$	100
$\clubsuit c/v$ - $4.26\clubsuit^2$	69
$(v/c)^2 \leq \mathit{SapsBestCVMean}$	53
$\clubsuit c/v$ - $4.26\clubsuit \leq \mathit{SapsBestCVMean}$	33

Feature Importance – Variable Ratio

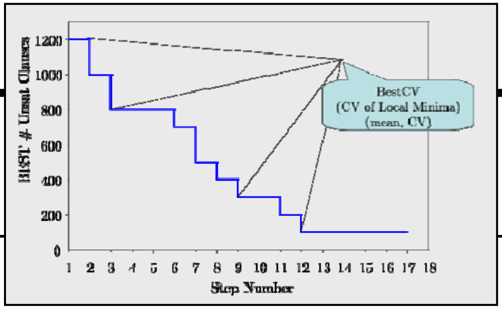
- **Subset selection** was used to identify features **sufficient** for achieving good performance
- As before, other (correlated) subsets could potentially achieve similar performance

Variable	Cost of Omission
$\clubsuit c/v - 4.26 \clubsuit$	100
$\clubsuit c/v - 4.26 \clubsuit^2$	69
$(v/c)^2 \leq SapsBestCVMean$	53
$\clubsuit c/v - 4.26 \clubsuit \leq SapsBestCVMean$	33

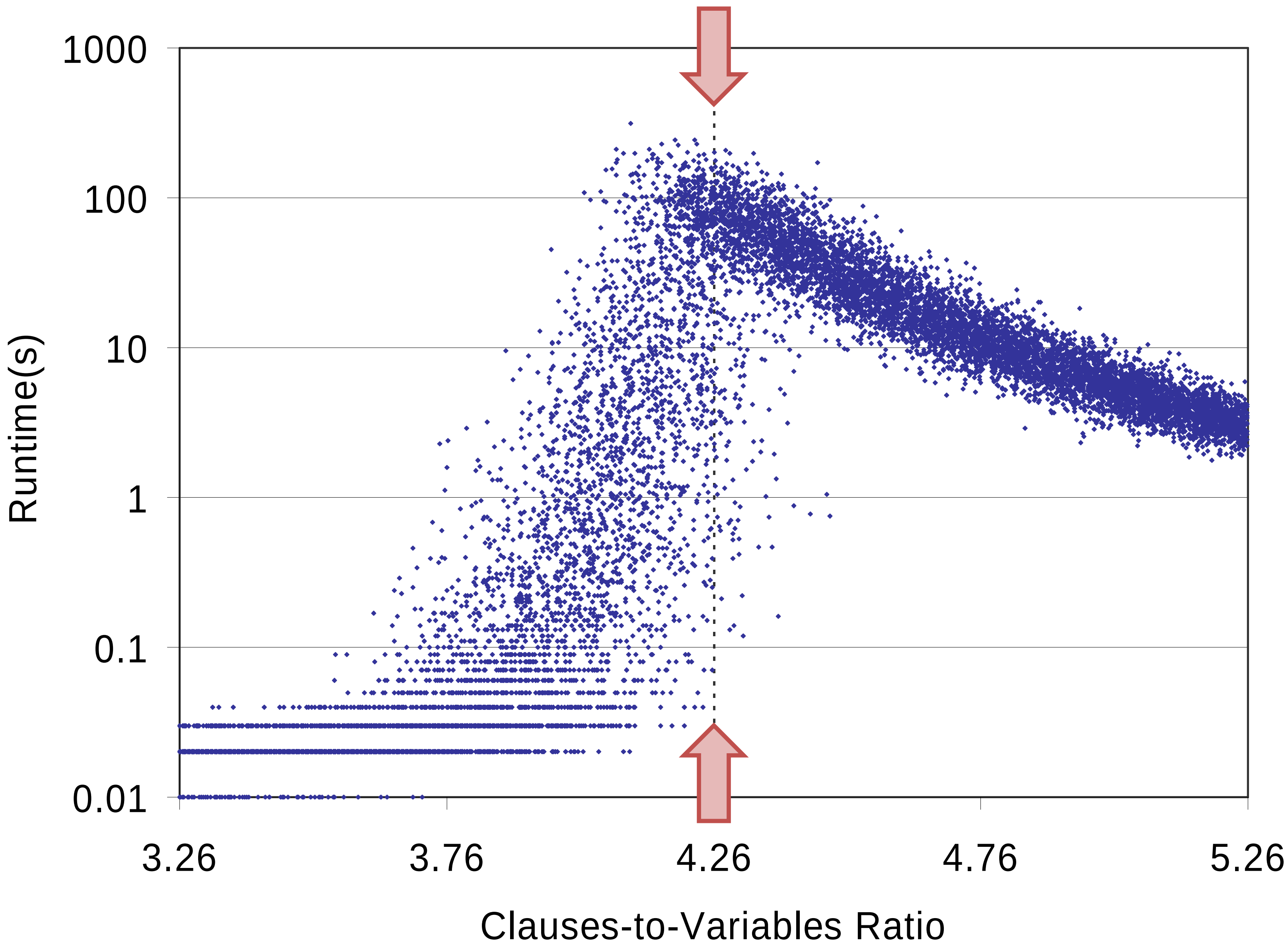
Feature Importance – Variable Ratio

- **Subset selection** was used to identify features **sufficient** for achieving good performance
- As before, other (correlated) subsets could potentially achieve similar performance

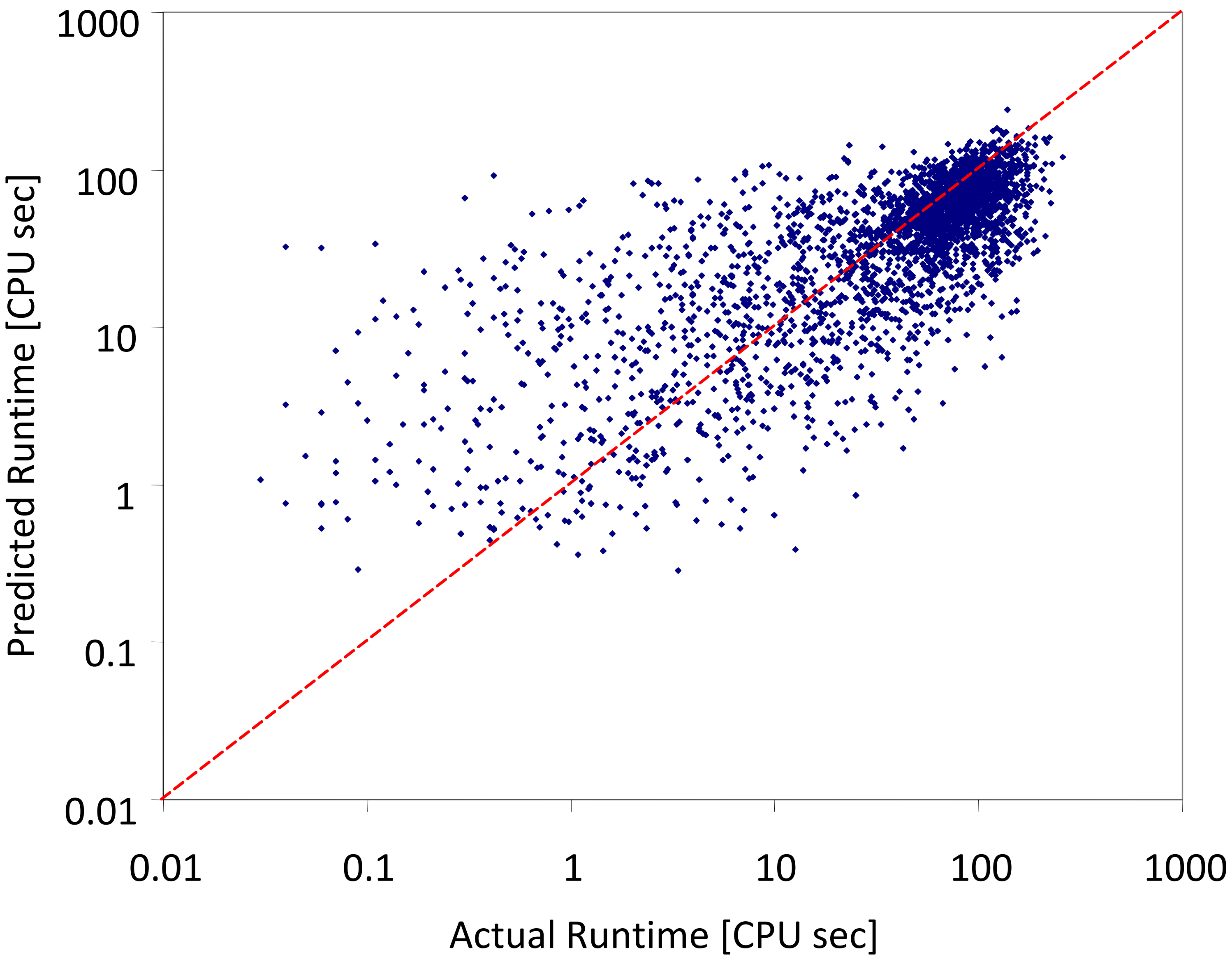
Variable	Cost of Omission
$\clubsuit c/v - 4.26 \clubsuit$	100
$\clubsuit c/v - 4.26 \clubsuit^2$	69
$(v/c)^2 \leq \mathit{SapsBestCVMean}$	53
$\clubsuit c/v - 4.26 \clubsuit \leq \mathit{SapsBestCVMean}$	33



Fixed Ratio Data



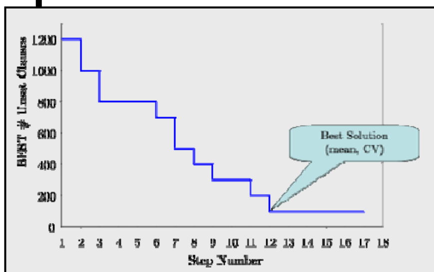
Fixed Ratio Prediction (Kcnfs)



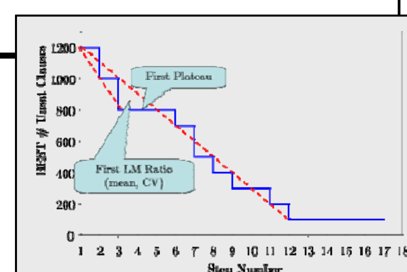
Feature Importance – Fixed Ratio

Variable	Cost of Omission
$SapsBestSolMean^2$	100
$SapsBestSolMean \leq MeanDPLLDepth$	74
$GsatBestSolCV \leq MeanDPLLDepth$	21
$VCGClauseMean \leq GsatFirstLMRatioMean$	9

Feature Importance – Fixed Ratio

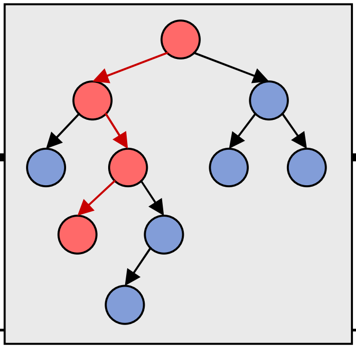


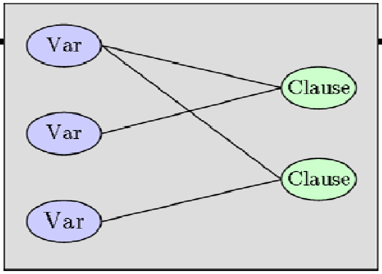
Variable	Cost of Omission
$SapsBestSolMean^2$	100
$SapsBestSolMean \leq MeanDPLLDepth$	74
$GsatBestSolCV \leq MeanDPLLDepth$	21
$VCGClauseMean \leq GsatFirstLMRatioMean$	9



Feature Importance – Fixed Ratio

Variable	Cost of Omission
$SapsBestSolMean^2$	100
$SapsBestSolMean \leq MeanDPLLDepth$	74
$GsatBestSolCV \leq MeanDPLLDepth$	21
$VCGClauseMean \leq GsatFirstLMRatioMean$	9



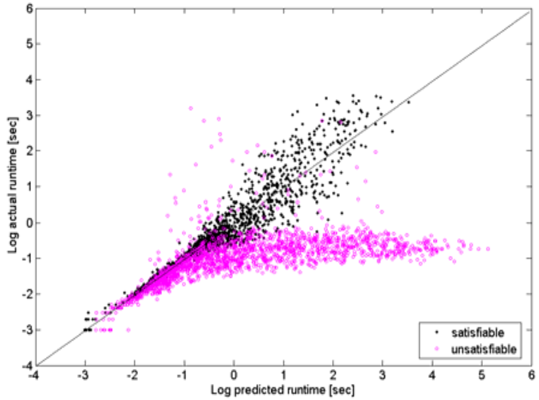


SAT vs. UNSAT

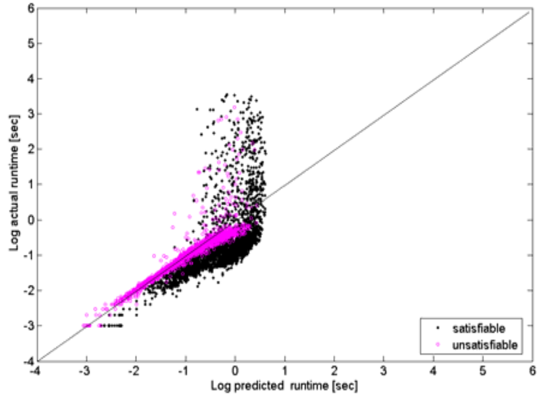
- **Training models separately** for SAT and UNSAT instances:
  - good models require **fewer features**
  - model **accuracy improves**
  - c/v no longer an important feature for VR data
  - Completely **different features** are useful for SAT than for UNSAT
- Feature importance on **SAT** instances:
  - **Local Search** features sufficient
    - 7 features for good VR model
    - 1 feature for good FR model (SAPSBestSolCV x SAPSAveImpMean)
  - If LS features omitted, **LP + DPLL search space** probing
- Feature importance on **UNSAT** instances:
  - **DPLL search space** probing
  - **Clause graph** features

Hierarchical Hardness Models

- We can leverage the fact that we can build strong "conditional hardness models" by combining them into a **hierarchical hardness model** [Xu, Hoos, Leyton-Brown, 2007]:
  1. Predict satisfiability status
  2. Use this prediction as a feature to combine the predictions of SAT-only and UNSAT-only models
- Not necessarily easy: SAT-only and UNSAT-only models can make **large errors when given wrong data**

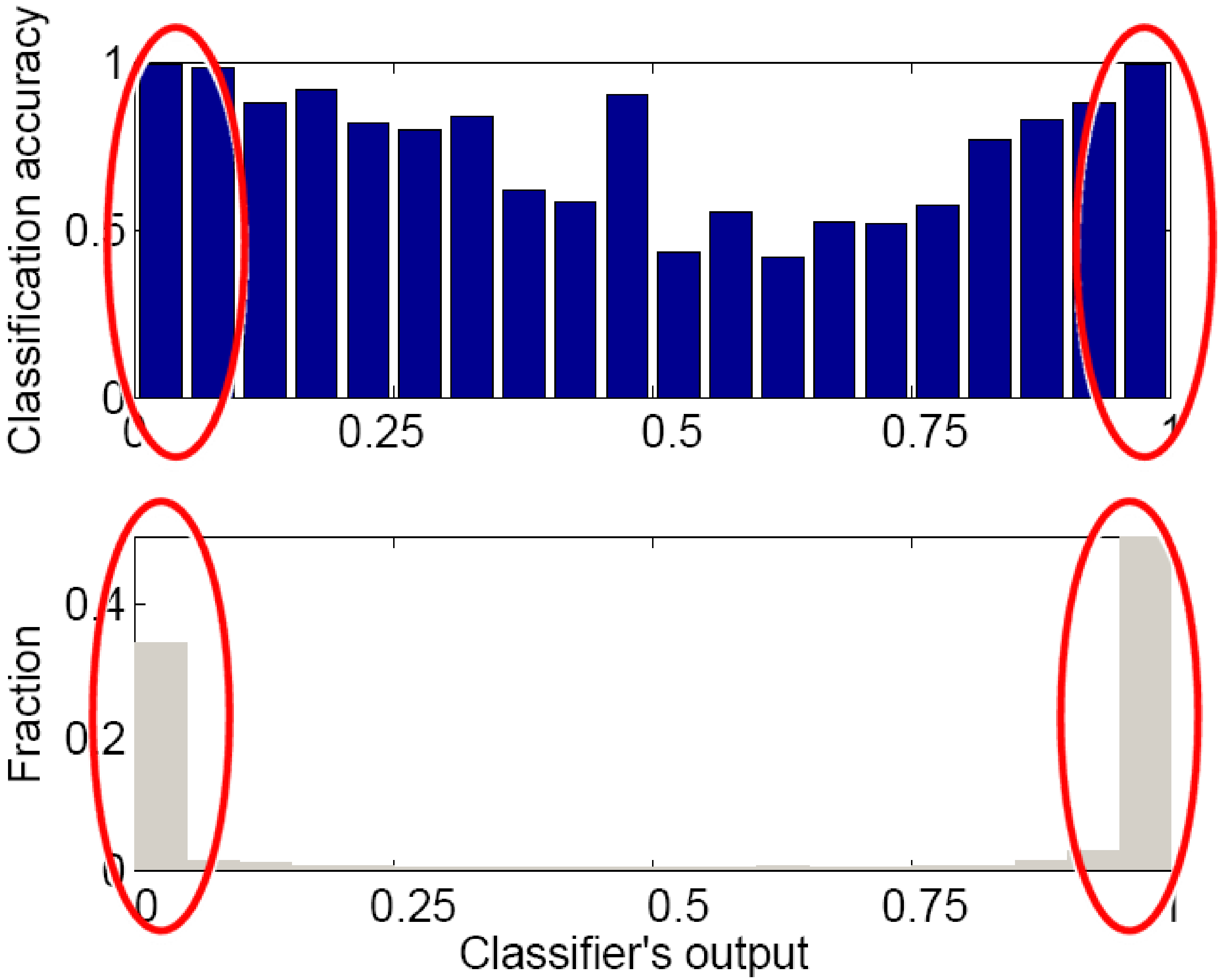


SAT-only



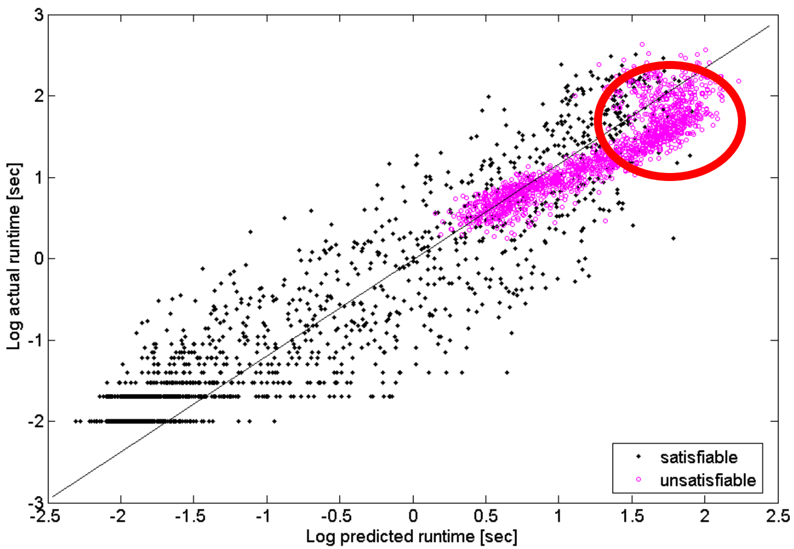
UNSAT-only

Predicting Satisfiability Status (fixed-ratio 3-SAT)

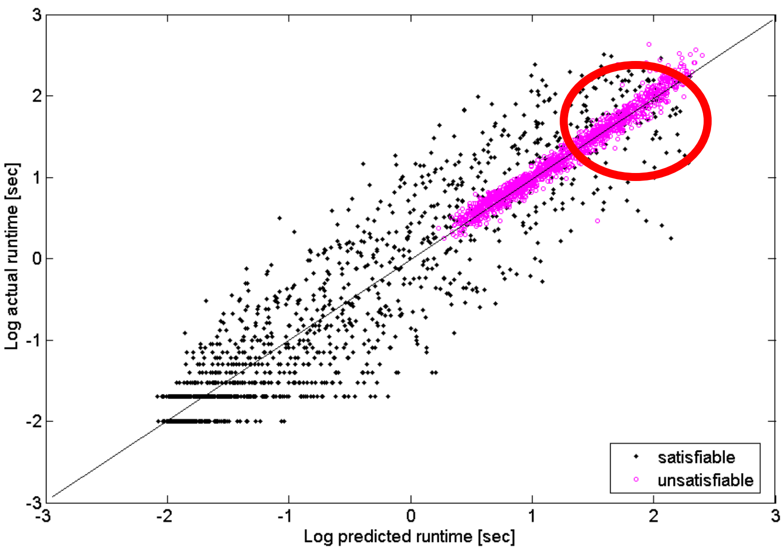


Example for Variable-Ratio 3-SAT (Solver: satz)

- Then we use a **mixture of experts** approach to learn our hierarchical hardness model
  - the **experts** are clamped to our SAT and UNSAT models
  - the **classifier's prediction** is a feature used to select the expert
  - the **model** is trained using EM

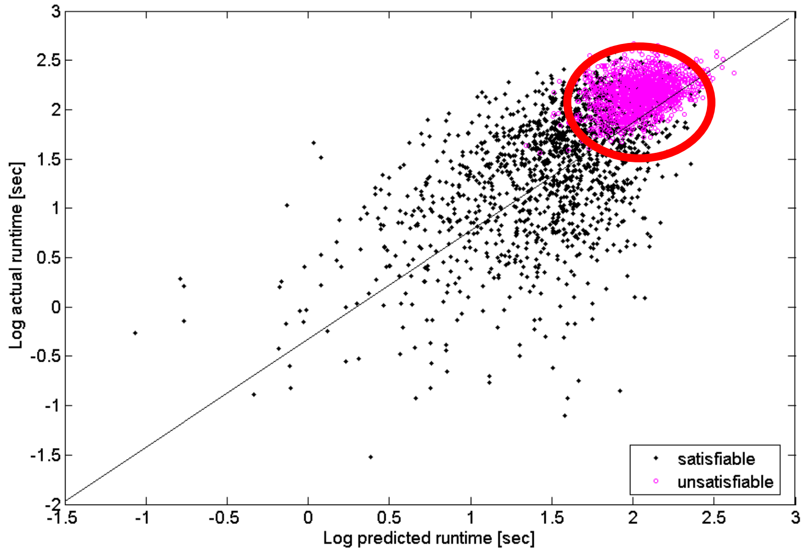


unconditional model

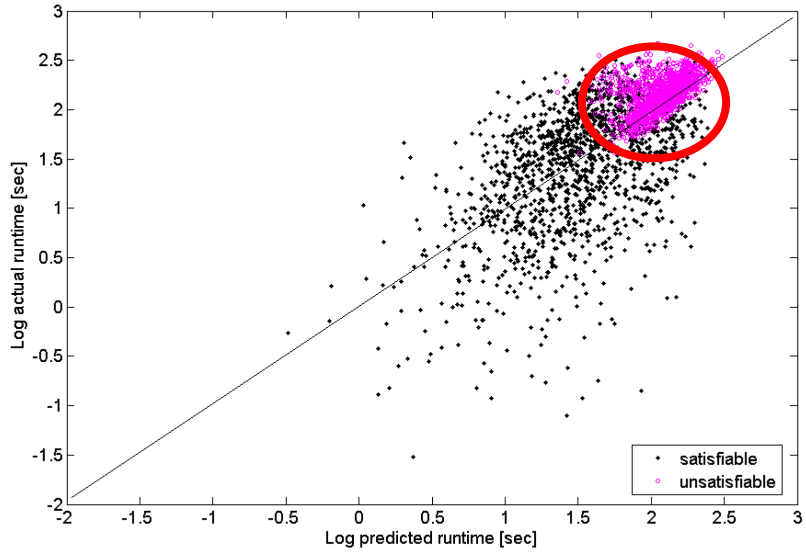


hierarchical model

Example for Fixed-Ratio 3-SAT (solver: satz)



unconditional model



hierarchical model

Dealing with Censored Data

- When runs can take weeks, some runs will have to be killed **before the algorithm terminates**
- This is called **censored data**. Three ways to handle it:
  - Drop all capped data
  - Label this data as having finished at cutoff
    - this is what we did in our combinatorial auction work
  - Censored sampling (from survival analysis)
- **Schmee & Hahn**'s algorithm [1979]

```
Repeat:
 1.  Evaluate EHM to estimate runtime for each capped
     instance, conditioning on the fact that
     the true runtime is more than the cutoff time
 2.  Build a new EHM based on these estimated runtimes
 Until no more changes in the model
```

Other Work on EHMs for SAT

- Building models for **structured SAT distributions**
  - we've had success with many other, more realistic distributions [Xu, Hoos, Leyton-Brown, 2007]; [Xu, Hutter, Hoos, Leyton-Brown, 2007]
  - I've just focused on uniform 3-SAT here to keep things simple
- Predicting runtime for **incomplete algorithms**
  - problem: runtime is not always the same on each instance!
  - solution: leverage probabilistic interpretation of regression; predict mean of runtime for given feature values [Hutter, Hamadi, Hoos, Leyton-Brown, 2006]
- Using models to **automatically tune algorithm parameters** in order to improve performance
  - considered this in past work [Hutter, Hamadi, Hoos, Leyton-Brown, 2006]
  - topic of active ongoing research [Hutter, Hoos, Leyton-Brown, Murphy]

IV. SATZILLA: AN ALGORITHM PORTFOLIO FOR SAT

[Nudelman, Devkar, Shoham, Leyton-Brown, Hoos, 2003]
[Nudelman, Devkar, Shoham, Leyton-Brown, Hoos, 2004]
[Xu, Hutter, Hoos, Leyton-Brown, 2007]
[Xu, Hutter, Hoos, Leyton-Brown, 2008]

some slides based on originals by Lin Xu

SATzilla

- There are many high performance SAT solvers, but **none is dominant**
- Instead of using a “winner-take-all” approach, the work I’ll describe here advocates building an **algorithm portfolio** based on empirical hardness models
- In particular, I’ll describe **SATzilla**:
  - an algorithm portfolio constructed from 19 state-of-the-art complete and incomplete SAT solvers
  - it won 5 medals at the 2007 SAT competition.

SATzilla Methodology (offline)

1. Identify a target **instance distribution**
2. Select a set of candidate **solvers**
3. Identify a set of instance **features**
4. On a training set, **compute features and solver runtimes**
5. Identify a set of "**presolvers**." Discard data that they can solve within a given cutoff time
6. Identify a "**backup solver**": the best on remaining data
7. Build an **empirical hardness model** for each solver from step (2)
8. Choose a subset of the solvers to **include in the portfolio**: those for which the best portfolio performance is achieved on new instances from a validation set

SATzilla Methodology (online)

9. Sequentially **run each presolver** until cutoff time
   - if the instance is solved, end
10. Compute **features**
    - if there's an error, run the backup solver
11. **Predict runtime** for each solver using the EHMs
12. **Run the algorithm** predicted to be best
    - if it crashes, etc., run the next-best algorithm

Solvers in SATzilla

- **SATzilla07**
  - the version we entered in the SAT competition
  - 7 complete solvers
  - SAPS, a local search algorithm as a pre-solver

- **SATzilla07^{+}**
  - The 7 complete solvers from SATzilla07
  - 8 new complete solvers from the 2007 SAT competition
  - 4 local search solvers from the 2007 SAT competition

Presolving

- Three **consequences of presolving**
  - Solve easy instances without feature computation overhead
  - Filter out easy instances and allow prediction models to focus more on hard instances
  - Increase runtime on instances not solved during presolving

- How to **select presolvers**
  - SATzilla07: manually
  - SATzilla07^{+}: automatically
    - Predefined set of presolvers and allowed cutoff times
    - Exhaustively search all possible combinations

Building Runtime Models

- Predict **performance score**
  - optimize for the quantity we actually care about
  - also makes it easier to add local search, which has infinite runtime on UNSAT instances
- We also used **censored sampling**
- SATzilla07:
  - Predict **runtime using HHM with two experts** (SAT/UNSAT)
- SATzilla07^{+}:
  - Predict **performance score using HHM with two experts** (SAT/UNSAT)
  - Predict **performance score using HHM with six experts** (3 categories × SAT/UNSAT)

2007 SAT Competition

- More than **40 solvers**
- Three categories of **instances**
  - Random
  - Handmade
  - Industrial
- Each category has three **events**
  - SAT
  - UNSAT
  - SAT+UNSAT
- Performance evaluated by a **scoring function** based on:
  - Solution purse (shared among solvers that solve the instance)
  - Speed purse (awarded to solvers based on solution time)
  - Series purse (shared among solvers that solve at least one inst/series)

SATzilla07 in 2007 SAT Competition

SAT 2007 competition

Organizing committee	Daniel Le Berre, Olivier Roussel and Laurent Simon
Judges	Ewald Speckenmeyer, Geoff Sutcliffe and Lintao Zhang
Benchmarks	random (tar.bz2 44MB), crafted (.tar, bz2 compressed files inside 175MB), industrial (.tar, bz2 compressed files inside, 556 MB)+ velev 's VLIW-SAT 4.0 and VLIW-UNSAT 2.0 + IBM benchmarks
Systems	All/Winners precompiled for linux (tgz, 25/10 MB). Source code (competition division only, tgz, -updated 11/7/07- 6MB).

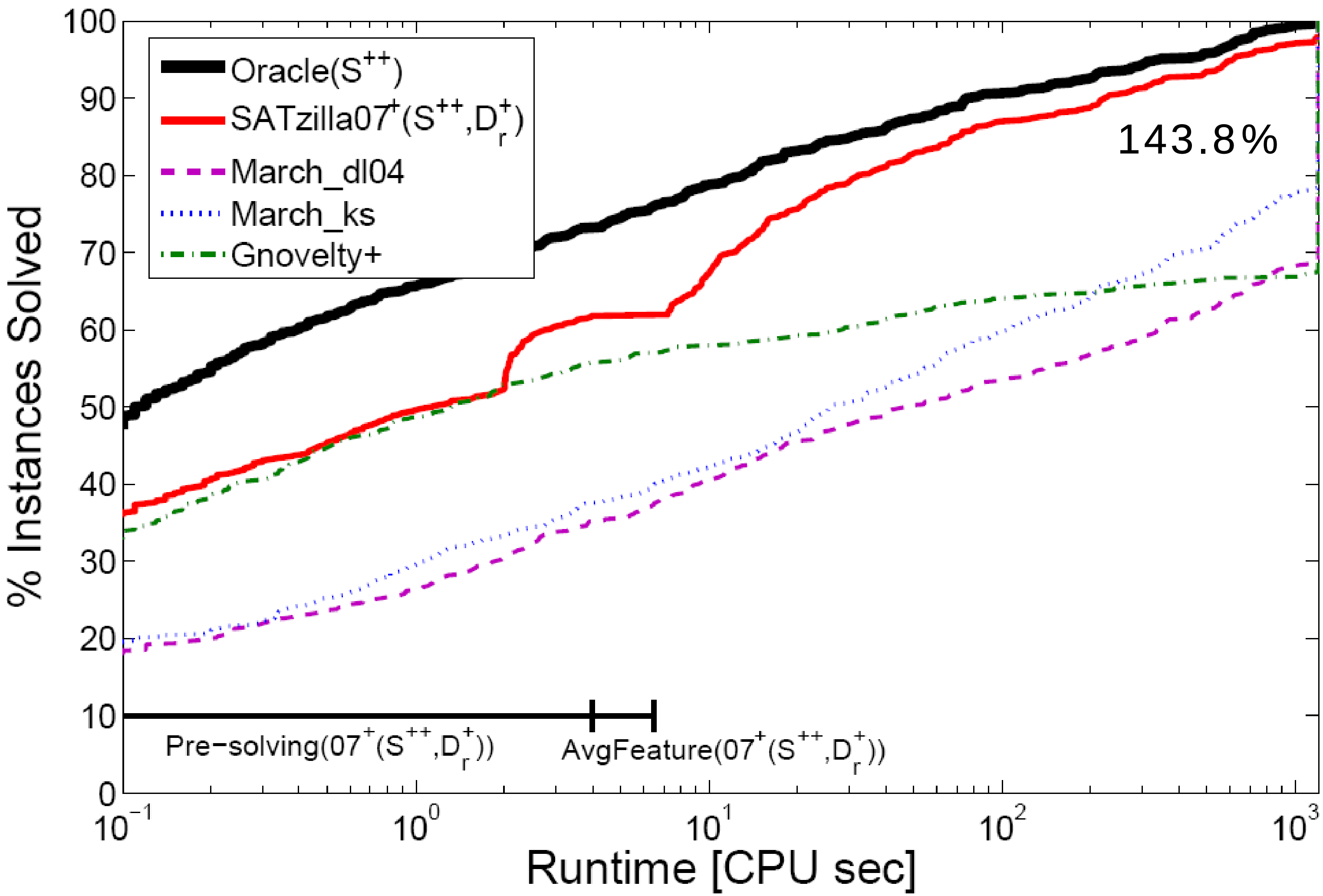
	Industrial			handmade			Random		
	Gold	Silver	Bronze	Gold	Silver	Bronze	Gold	Silver	Bronze
SAT+UNSAT	Rsat	Picosat	Minisat	SATzilla CRAFTED	Minisat	MXC	SATzilla RANDOM	March KS	KCNFS 2004
SAT	Picosat	Rsat	Minisat	March KS	SATzilla CRAFTED	Minisat	gnovelty+	adaptg2wsat0	adaptg2wsat+
UNSAT	Rsat	Minisat	TiniSatELite	SATzilla CRAFTED	TTS	Minisat	March KS	KCNFS 2004	SATzilla RANDOM

SATzilla for Random

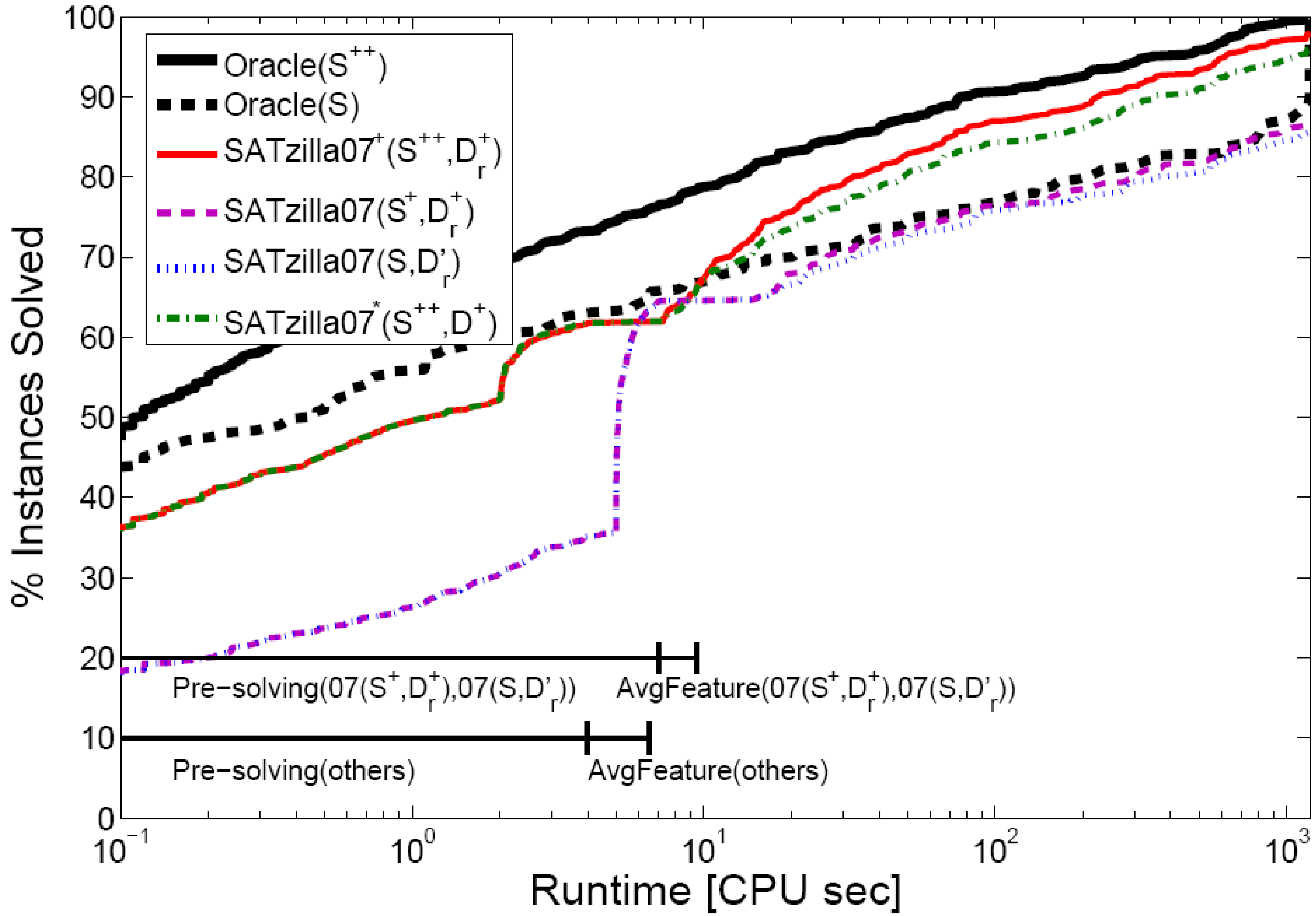
SATzilla version	Pre-Solvers (time)	Component solvers
SATzilla07(S,D'_r)	March_dl04(5); SAPS(2)	Kcnfs06, March_dl04, Rsat 1.03
SATzilla07(S^+,D_r^+)	March_dl04(5); SAPS(2)	Kcnfs06, March_dl04, March_ks, Minisat07
SATzilla07$^+$(S^{++},D_r^+)	SAPS(2); Kcnfs06(2)	Kcnfs06, March_ks, Minisat07, Ranov, Ag2wsat+, Gnovelty+

Solver	Avg. runtime [s]	Solved [%]	Performance score
Kcnfs04	852	32.1	38309
March_ks	**351**	**78.4**	113666
Ag2wsat0	479	62.0	119919
Ag2wsat+	510	59.1	110218
Gnovelty+	410	67.4	**131703**
SATzilla07(S,D'_r)	231	85.4	— (86.6%)
SATzilla07(S^+,D_r^+)	218	86.5	— (88.7%)
SATzilla07$^+$(S^{++},D_r^+)	**84**	**97.8**	**189436 (143.8%)**
SATzilla07*(S^{++},D^+)	113	95.8	— (137.8%)

Comparing with State of the Art



Comparing Different SATzilla Versions

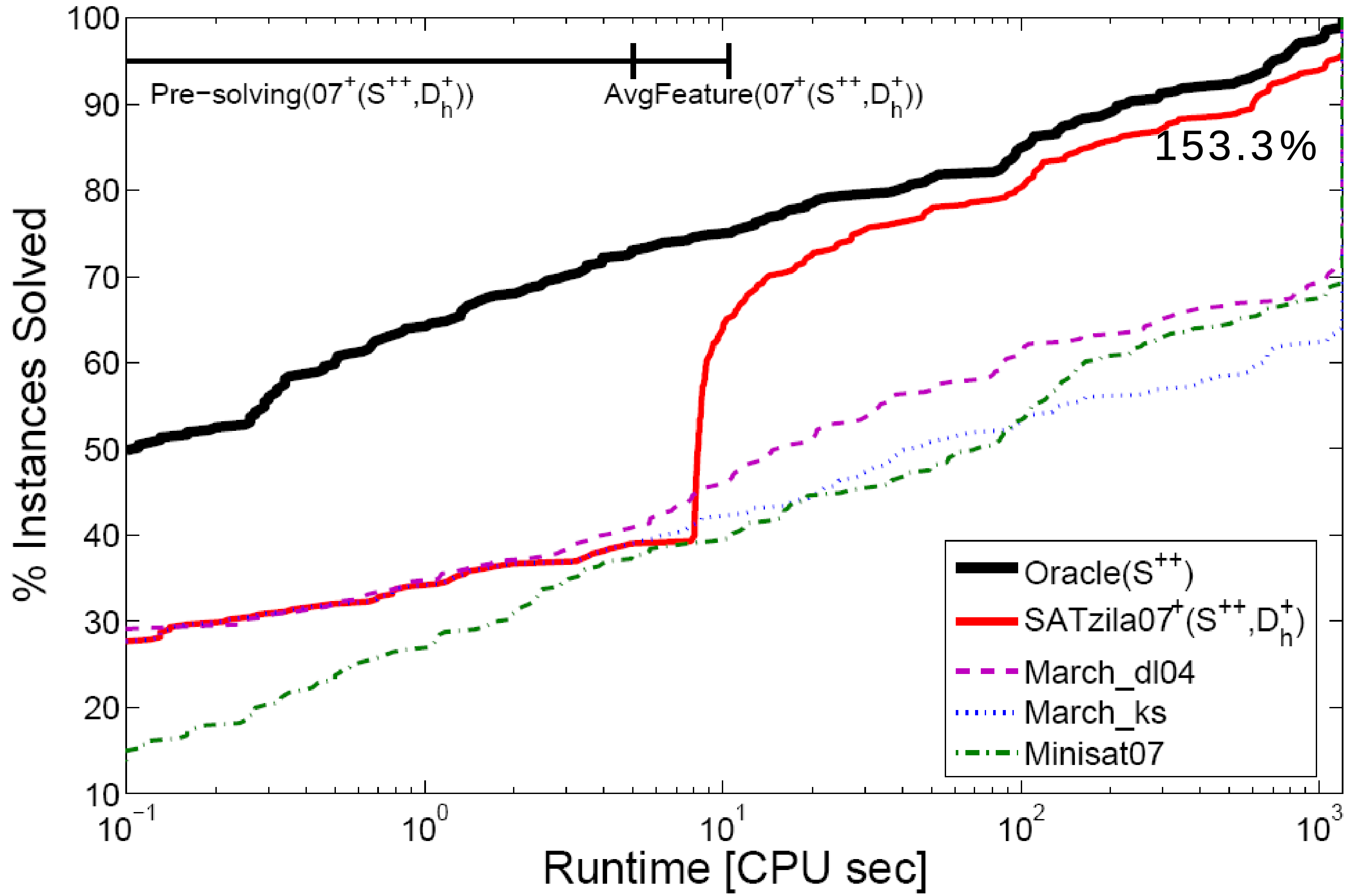


Detailed Analysis of SATzilla07$^+$

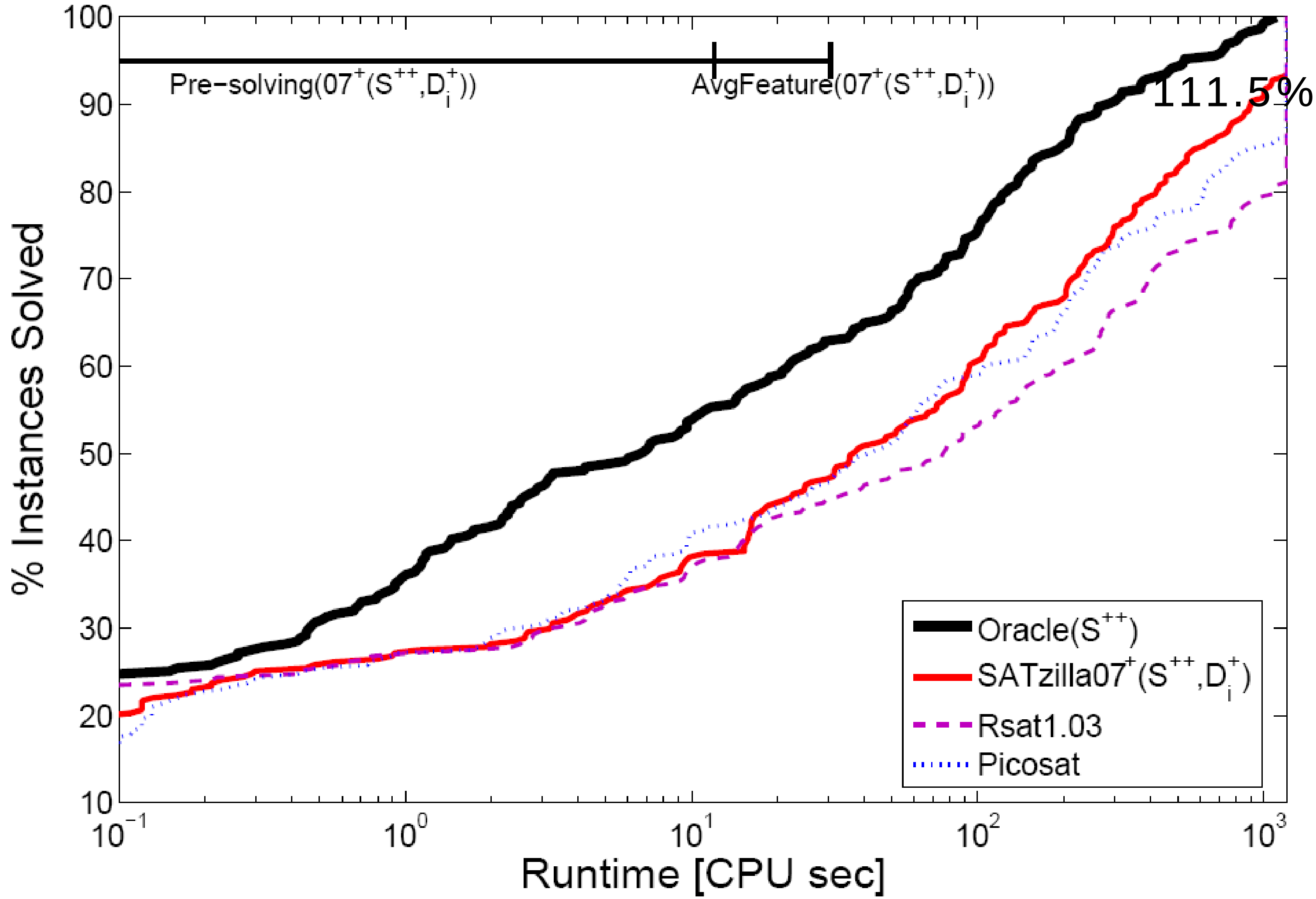
Pre-Solver (Pre-Time)	Solved [%]	Avg. Runtime [CPU sec]
SAPS(2)	52.2	1.1
March_dl04(2)	9.6	1.68

Selected Solver	Selected [%]	Success [%]	Avg. Runtime [CPU sec]
March_dl04	34.8	96.2	294.8
Gnovelty+	28.8	93.9	143.6
March_ks	23.9	92.6	213.3
Minisat07	4.4	100	61.0
Ranov	4.0	100	6.9
Ag2wsat+	4.0	77.8	357.9

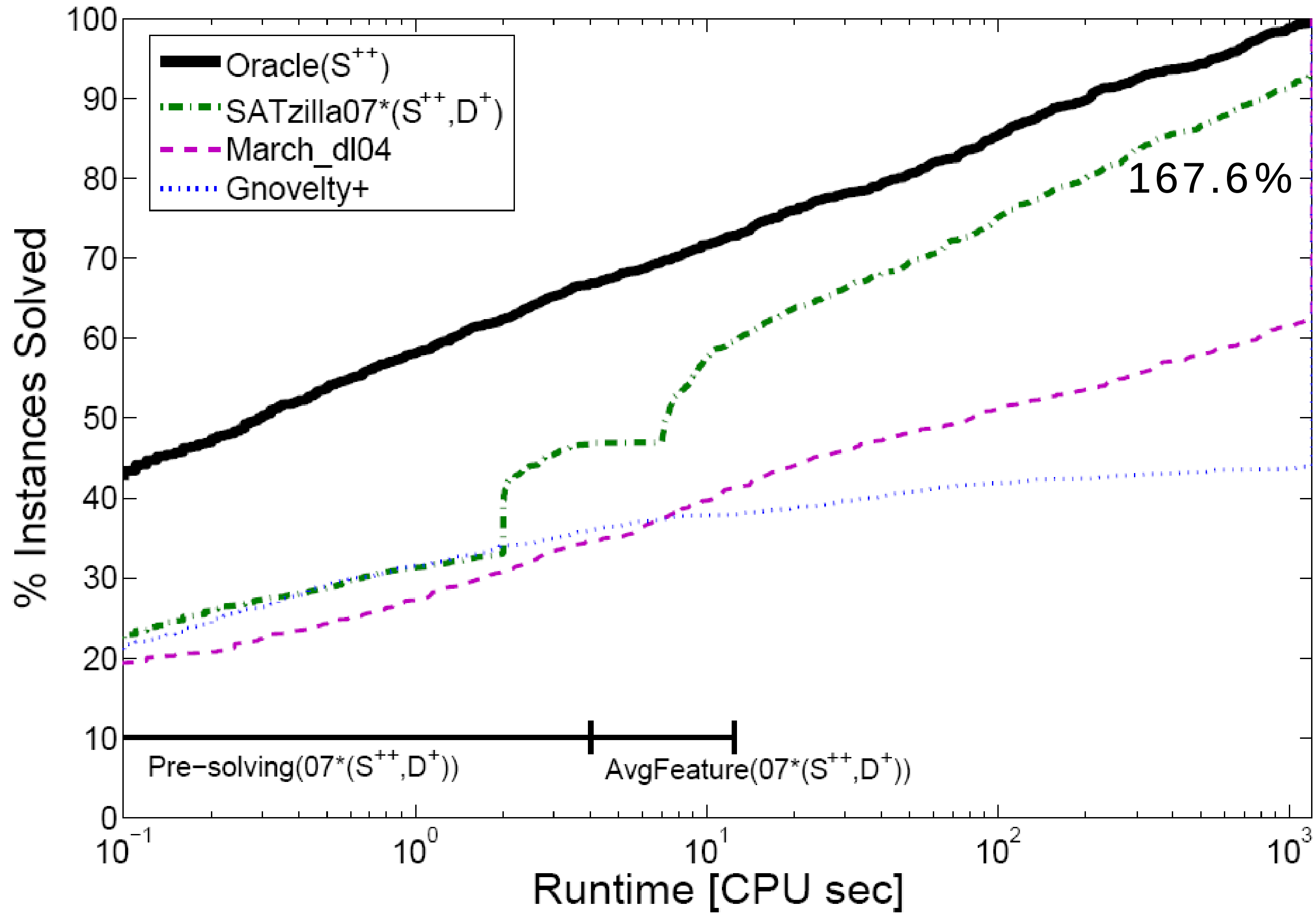
SATzilla for Handmade



SATzilla for Industrial



SATzilla for All (R+H+I)



Conclusions

- We've looked at how **empirical hardness models** can be used to tackle hard computational problems
- We began with **combinatorial auctions**, and looked at
  - constructing models
  - interpreting models via subset selection
  - building algorithm portfolios
  - making instance distributions harder
- Then we switched to **satisfiability**, and considered
  - building and interpreting models
  - predicting satisfiability status and building hierarchical models
  - SATzilla, a high-performance algorithm portfolio
- Overall, it's our experience that EHMs work for a wide variety of problems. **Why not try yours?**

Thanks for your attention!

I'd like again to acknowledge the co-authors who contributed to the work I've discussed today:

Galen Andrew
Alex Devar
Youssef Hamadi
Holger Hoos
Frank Hutter
James McFadden
Eugene Nudelman
Mark Pearson
Yoav Shoham
Lin Xu