Performance Prediction and Automated Tuning of Randomized and Parametric Algorithms

Frank Hutter[1], Youssef Hamadi[2], Holger Hoos[1], and Kevin Leyton-Brown[1]

[1]University of British Columbia, Vancouver, Canada
[2]Microsoft Research Cambridge, UK

Motivation: Performance Prediction

- Useful for research in algorithms
  - What makes problems hard?
  - Constructing hard benchmarks
  - Constructing algorithm portfolios (satzilla)
  - Algorithm design
- Newer applications
  - Optimal restart strategies
    (see previous talk by Gagliolo et al.)
  - Automatic parameter tuning (this talk)

Motivation: Automatic tuning

- Tuning parameters is a pain
  - Many parameters → combinatorially many configurations
  - About 50% of development time can be spent tuning parameters
- Examples of parameters
  - Tree Search: variable/value heuristics, propagation, restarts, …
  - Local Search: noise, tabu length, strength of escape moves, …
  - CP: modelling parameters + algorithm choice + algo params
- Idea: automate tuning with methods from AI
  - More scientific approach
  - More powerful: e.g. automatic per instance tuning
  - Algorithm developers can focus on more interesting problems

Related work

- Performance Prediction

  [Lobjois and Lemaître, ’98, Horvitz et. al ’01,
  Leyton-Brown, Nudelman et al. ’02 & ’04, Gagliolo & Schmidhuber ’06]

- Automatic Tuning
  - Best fixed parameter setting for instance set
    [Birattari et al. ’02, Hutter ’04, Adenso-Diaz & Laguna ’05]
  - Best fixed setting for each instance
    [Patterson & Kautz ’02]
  - Changing search strategy during the search
    [Battiti et al, ’05, Lagoudakis & Littman, ’01/’02, Carchrae & Beck ’05]

Overview

- Previous work on empirical hardness models
  [Leyton-Brown, Nudelman et al. '02 & '04]
- EH models for randomized algorithms
- EH models for parametric algorithms
- Automatic tuning based on these
- Ongoing Work and Conclusions

Empirical hardness models: basics

- Training: Given a set of t instances $inst_1,\ldots,inst_t$
  - For each instance $inst_i$
    - Compute instance features $\mathbf{x_i} = (x_{i1},\ldots,x_{im})$
    - Run algorithm and record its runtime y_i
  - Learn function f: features $\longrightarrow$ runtime, such that $y_{\mathbf{i}} \approx f(\mathbf{x}_i)$ for $i=1,\ldots,t$
- Test / Practical use: Given a new instance $inst_{t+1}$
  - Compute features $\mathbf{x_{t+1}}$
  - Predict runtime $y_{t+1} = f(\mathbf{x_{t+1}})$

Which instance features?

- Features should be computable in polytime
  - Basic properties, e.g. #vars, #clauses, ratio
  - Graph-based characterics
  - Local search and DPLL probes
- Combine features to form more expressive basis functions $\phi = (\phi_1,\ldots,\phi_q)$
  - Can be arbitrary combinations of the features $x_1,\ldots,x_m$
- Basis functions used for SAT in [Nudelman et al. ’04]
  - 91 original features: x_i
  - Pairwise products of features: $x_i * x_j$
  - Feature selection to pick best basis functions

How to learn function f: features → runtime?

- Runtimes can vary by orders of magnitude
  - Need to pick an appropriate model
  - Log-transform the output
    e.g. runtime is 10^3 sec ⇔ $y_i = 3$
- Simple functions show good performance
  - Linear in the basis functions: $y_i \approx f(\phi_i) = \phi_i * \mathbf{w}^T$
    - Learning: fit the weights **w**
      (ridge regression: $\mathbf{w} = (\lambda + \Phi^T \Phi)^{-1} \Phi^T y$)
  - Gaussian Processes didn't improve accuracy

Overview

- Previous work on empirical hardness models [Leyton-Brown, Nudelman et al. '02 & '04]
- EH models for randomized algorithms
- EH models for parametric algorithms
- Automatic tuning based on these
- Ongoing Work and Conclusions

EH models for randomized algorithms

- We have incomplete, randomized local search algorithms
  - Can this same approach still predict the run-time ? Yes!
- Algorithms are incomplete (local search)
  - Train and test on satisfiable instances only
- Randomized
  - Ultimately, want to predict entire run-time distribution (RTDs)
  - For our algorithms, RTDs are typically exponential
  - Can be characterized by a single sufficient statistic (e.g. median run-time)

EH models: basics → sufficient stats for RTD

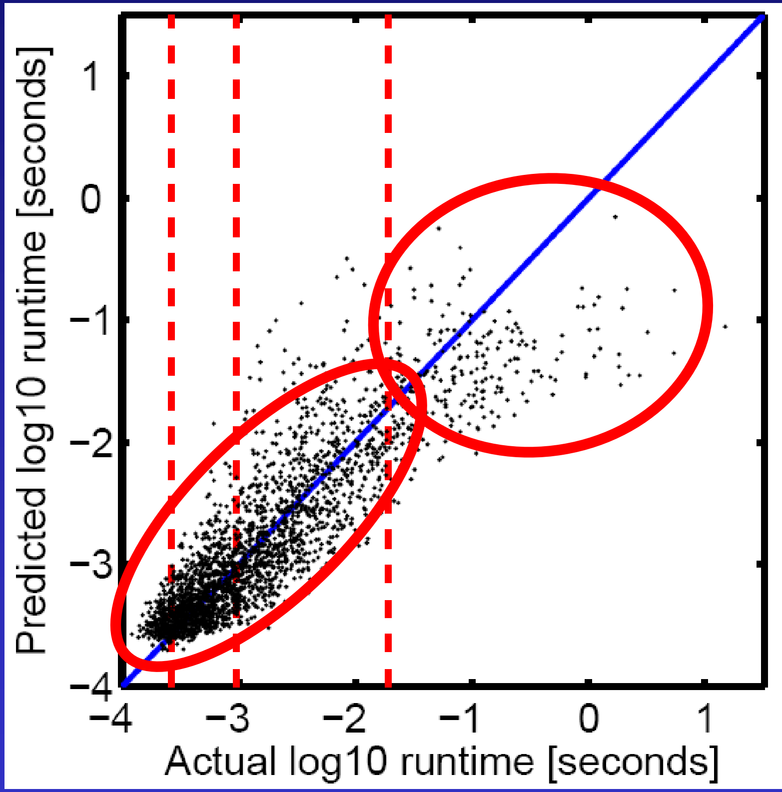
- Training: Given a set of t instances $inst_1, \ldots, inst_t$
  - For each instance $inst_i$
    - Compute instance features $\mathbf{x_i} = (x_{i1}, \ldots, x_{im})$
    - Compute basis functions $\boldsymbol{\phi}_\mathbf{i} = (\phi_{i1}, \ldots, \phi_{ik})$
    - Run algorithm and record its runtime y_i
  - Learn function f: basis functions → runtime, such that $y_\mathbf{i} \approx f(\boldsymbol{\phi}_\mathbf{i})$ for i=1,…,t
- Test / Practical use: Given a new instance $inst_{t+1}$
  - Compute features $\mathbf{x_{t+1}}$
  - Compute basis functions $\boldsymbol{\phi}_\mathbf{t+1} = (\phi_{\mathbf{t+1},1}, \ldots, \phi_{\mathbf{t+1},k})$
  - Predict runtime $y_{t+1} = f(\boldsymbol{\phi}_\mathbf{t+1})$

EH models: basics $\longrightarrow$ sufficient stats for RTD

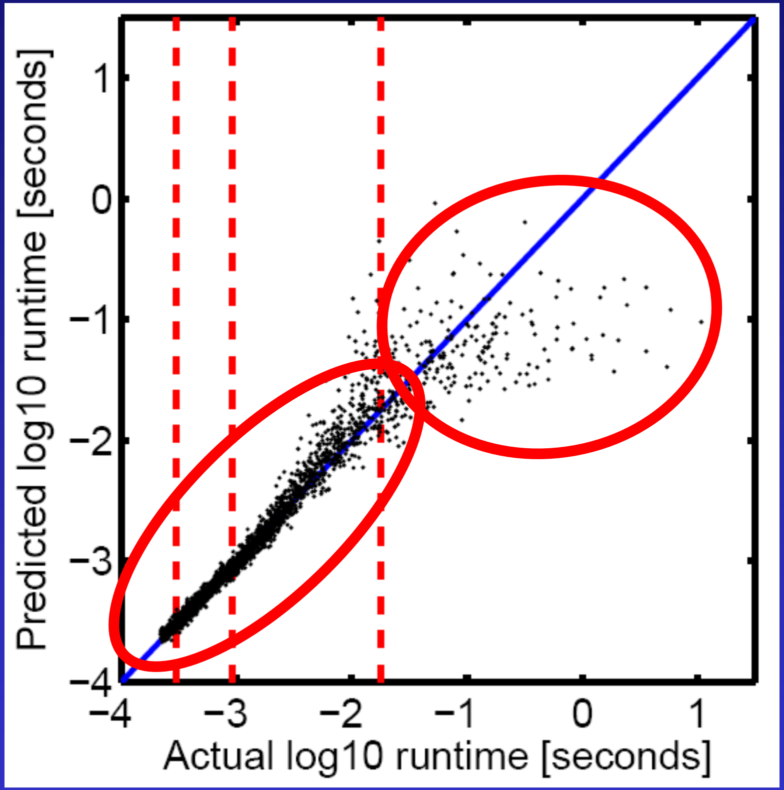
- Training: Given a set of t instances $inst_1, \ldots, inst_t$
  - For each instance $inst_i$
    - Compute instance features $\mathbf{x_i} = (x_{i1}, \ldots, x_{im})$
    - Compute basis functions $\boldsymbol{\phi}_\mathbf{i} = (\phi_{i1}, \ldots, \phi_{ik})$
    - Run algorithm multiple times and record its runtimes $y_i^1, \ldots, y_i^k$
    - Fit sufficient statistics s_i for distribution from $y_i^1, \ldots, y_i^k$
  - Learn function f: basis functions $\longrightarrow$ sufficient statistics, such that $s_\mathbf{i} \approx f(\boldsymbol{\phi}_\mathbf{i})$ for $i=1,\ldots,t$
- Test / Practical use: Given a new instance $inst_{t+1}$
  - Compute features $\mathbf{x_{t+1}}$
  - Compute basis functions $\boldsymbol{\phi}_\mathbf{t+1} = (\phi_{\mathbf{t+1},1}, \ldots, \phi_{\mathbf{t+1},k})$
  - Predict sufficient statistics $s_{t+1} = f(\boldsymbol{\phi}_\mathbf{t+1})$

Predicting median run-time

Median runtime of Novelty$^+$ on CV-var



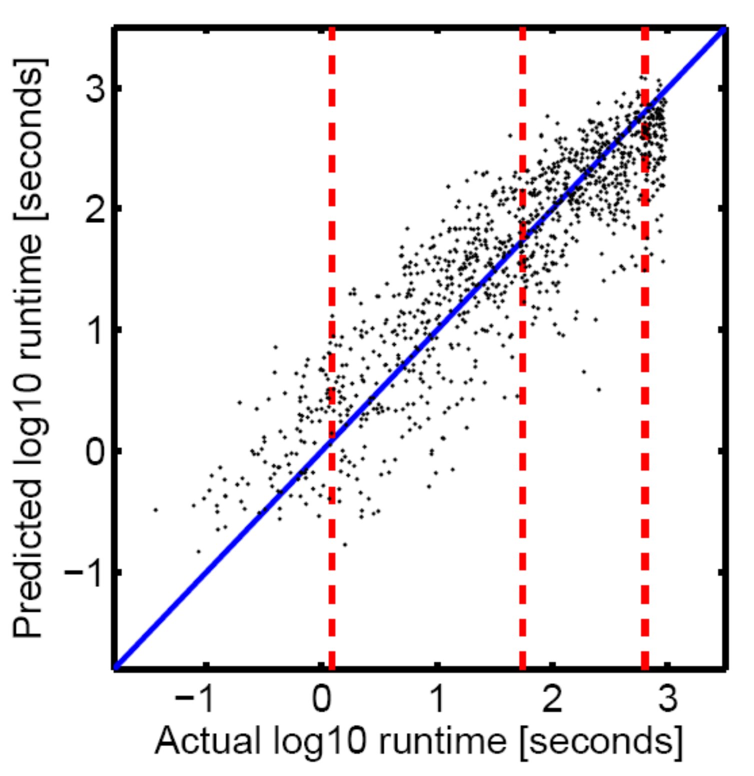
Prediction based on single runs



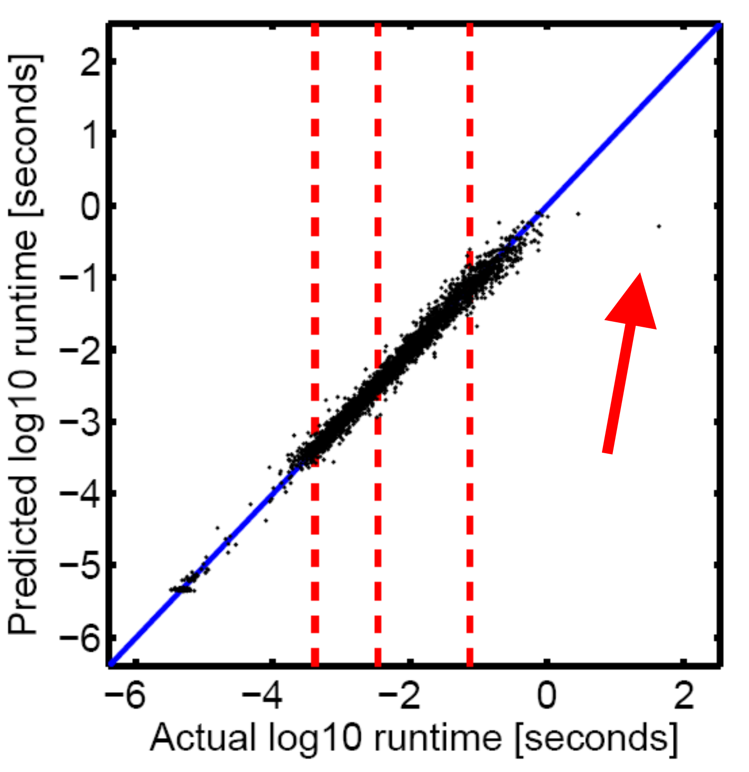
Prediction based on 100 runs

Structured instances

Median runtime predictions based on 10 runs

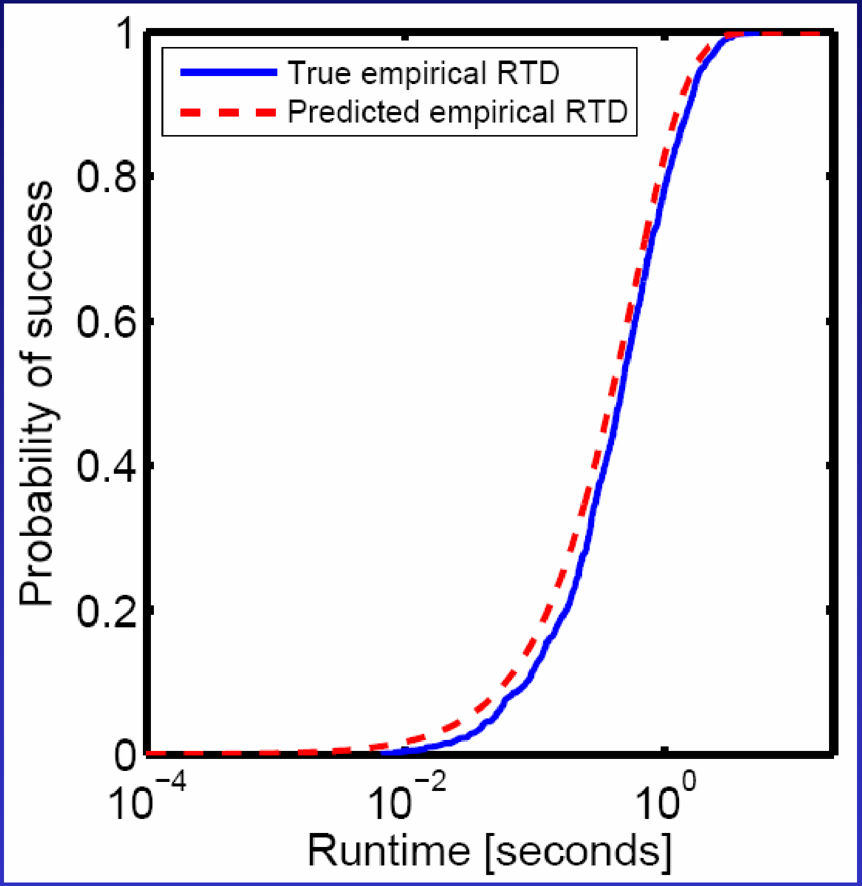


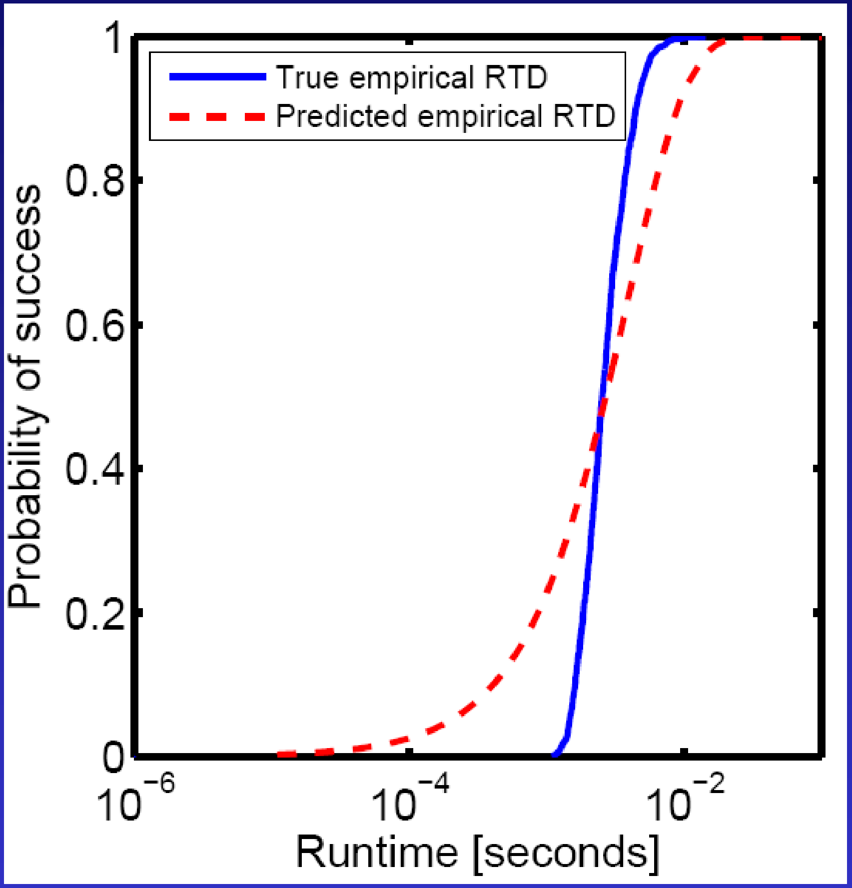
Novelty$^+$ on SW-GCP



SAPS on QWH

Predicting run-time distributions





RTD of SAPS on $q_{0.75}$ instance of QWH

RTD of SAPS on $q_{0.25}$ instance of QWH

Overview

- Previous work on empirical hardness models [Leyton-Brown, Nudelman et al. ’02 & ’04]
- EH models for randomized algorithms
- EH models for parametric algorithms
- Automatic tuning based on these
- Ongoing Work and Conclusions

EH models: basics $\rightarrow$ parametric algos

- Training: Given a set of t instances $inst_1, \ldots, inst_t$
  - For each instance $inst_i$
    - Compute instance features $\mathbf{x_i} = (x_{i1}, \ldots, x_{im})$

      Compute basis functions $\boldsymbol{\phi}_\mathbf{i} = \boldsymbol{\phi}(\mathbf{x_i})$

    - Run algorithm and record its runtime y_i
  - Learn function f: basis functions $\rightarrow$ runtime, such that $y_\mathbf{i} \approx f(\boldsymbol{\phi}_\mathbf{i})$ for $i=1,\ldots,t$
- Test / Practical use: Given a new instance $inst_{t+1}$
  - Compute features $\mathbf{x_{t+1}}$

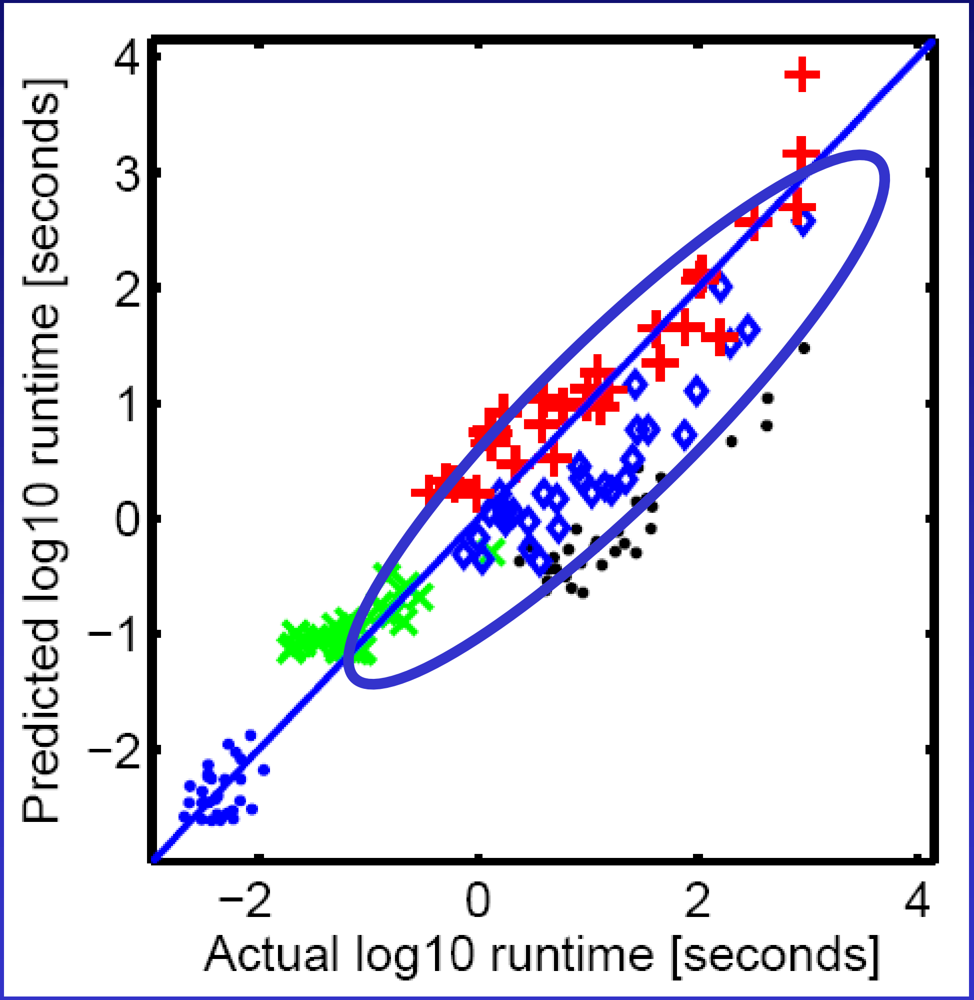
    Compute basis functions $\boldsymbol{\phi}_\mathbf{t+1} = \boldsymbol{\phi}(\mathbf{x_{t+1}})$
    Predict runtime $y_{t+1} = f(\boldsymbol{\phi}_\mathbf{t+1})$

EH models: basics → parametric algos

- Training: Given a set of t instances $inst_1, \ldots, inst_t$
  - For each instance $inst_i$
    - Compute instance features $\mathbf{x_i} = (x_{i1}, \ldots, x_{im})$
    - For parameter settings $p_i^1, \ldots, p_i^{n_i}$:
      Compute basis functions $\phi_i^j = \phi(\mathbf{x_i}, p_i^j)$ of features <u>and</u> parameter settings (quadratic expansion of params, multiplied by instance features)
    - Run algorithm with each setting p_i^j and record its runtime y_i^j
  - Learn function f: basis functions → runtime,
    such that $y_i^j \approx f(\phi_i^j)$ for i=1,…,t
- Test / Practical use: Given a new instance $inst_{t+1}$
  - Compute features $\mathbf{x_{t+1}}$
  - For each parameter setting p^j of interest,
    Compute basis functions $\phi_{t+1}^j = \phi(\mathbf{x_{t+1}}, p^j)$
    Predict runtime $y_{t+1}^j = f(\phi_{t+1}^j)$

Predicting SAPS with different settings

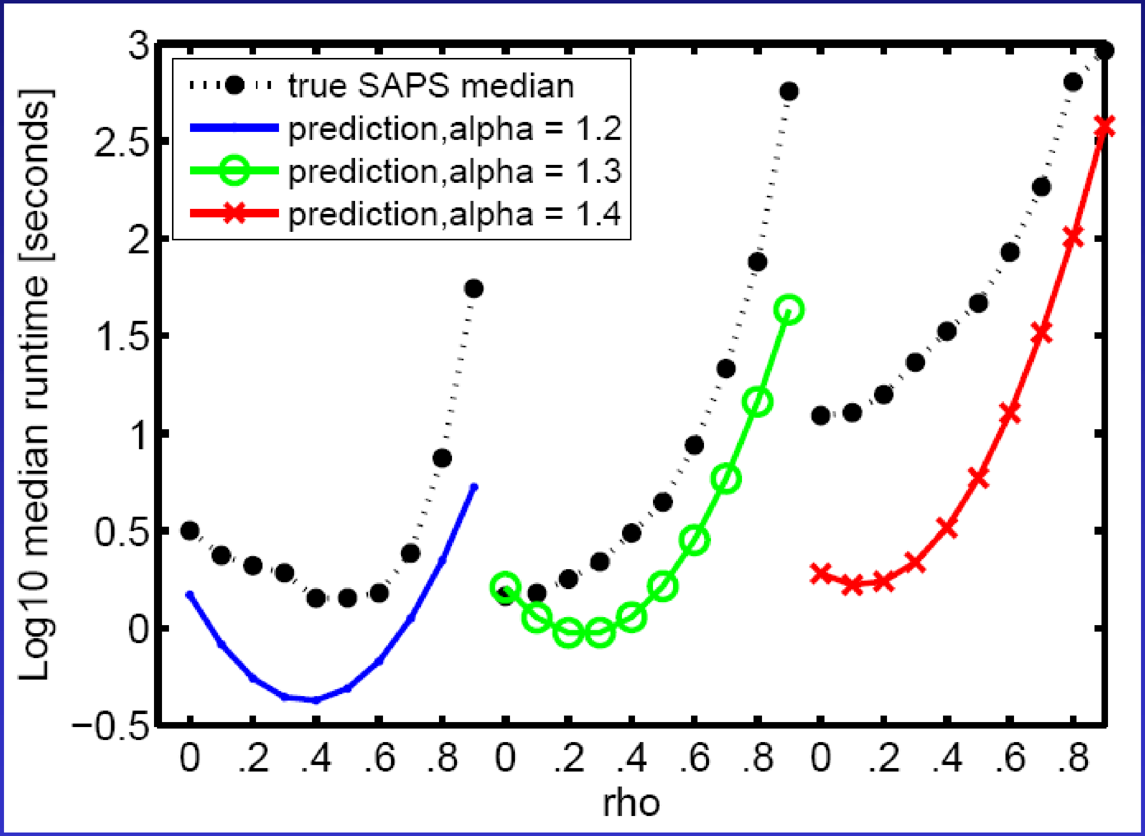
- Train and test with 30 different parameter settings on QWH
- Show 5 test instances, each with different symbol
  - Easiest
  - 25% quantile
  - Median
  - 75% quantile
  - Hardest
- More variation in harder instances



One instance in detail

(blue diamonds in previous figure)

- Note: this is a projection from 40-dimensional joint feature/parameter space
- Relative relationship predicted well



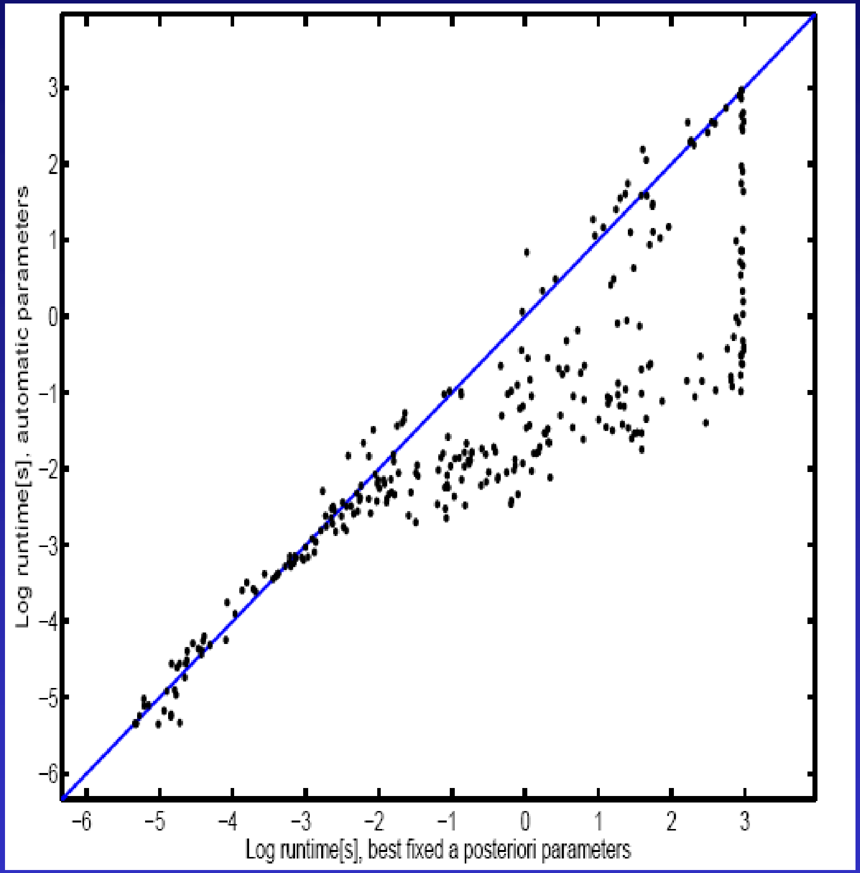
Automated parameter setting: results

Algo	Data Set	Speedup over default params	Speedup over best fixed params for data set
Nov$^+$	unstructured	0.90	0.90
Nov$^+$	structured	257	0.94
Nov$^+$	mixed	15	10
SAPS	unstructured	2.9	1.05
SAPS	structured	2.3	0.98
SAPS	mixed	2.31	1

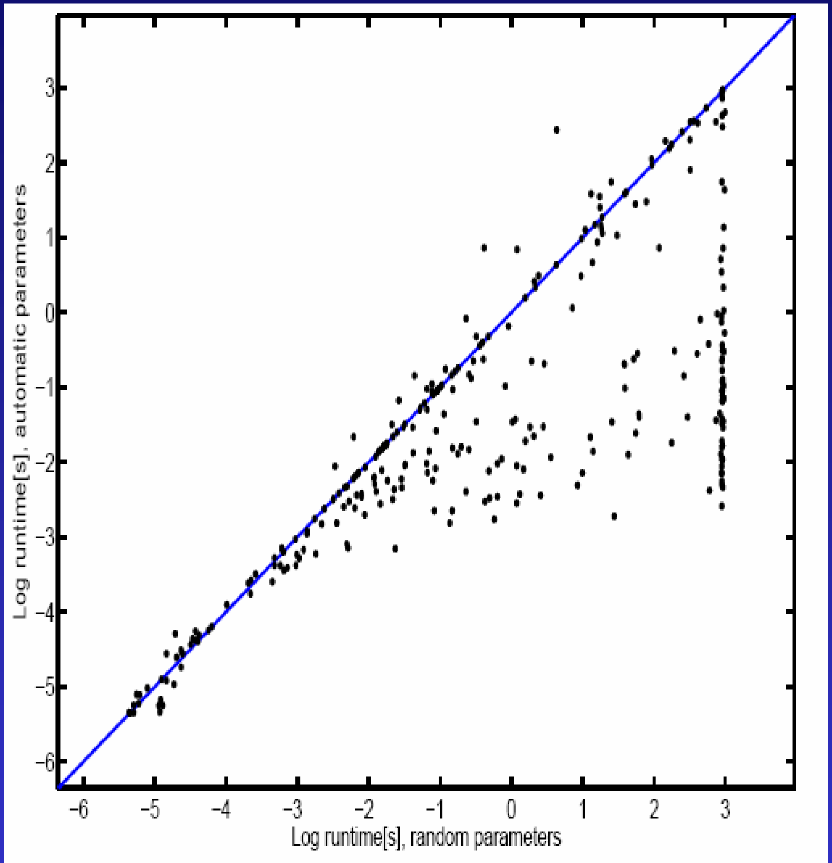
Not the best algorithm to tune ;-)

Do you have one?

Results for Novelty^{+} on Mixed



Compared to best fixed parameters



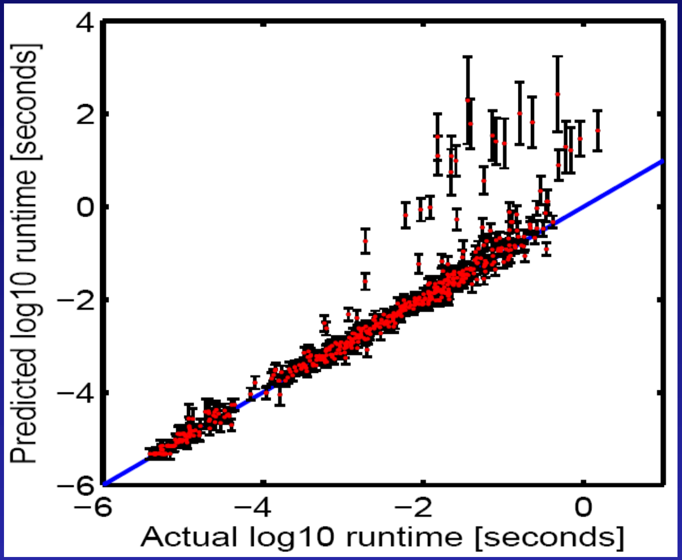
Compared to random parameters

Overview

- Previous work on empirical hardness models
  [Leyton-Brown, Nudelman et al. ’02 & ’04]
- EH models for randomized algorithms
- EH models for parametric algorithms
- Automatic tuning based on these
- Ongoing Work and Conclusions

Ongoing work

- Uncertainty estimates
  - Bayesian linear regression vs. Gaussian processes
  - GPs are better in predicting uncertainty



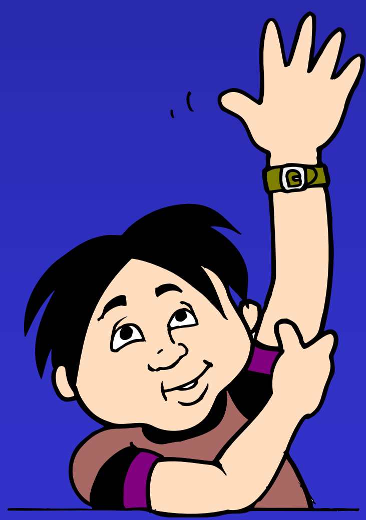
- Active Learning
  - For many problems, cannot try all parameter combinations
  - Dynamically choose best parameter configurations to train on
- Want to try more problem domains (do you have one?)
  - Complete parametric SAT solvers
  - Parametric solvers for other domains (need features)
  - Optimization algorithms

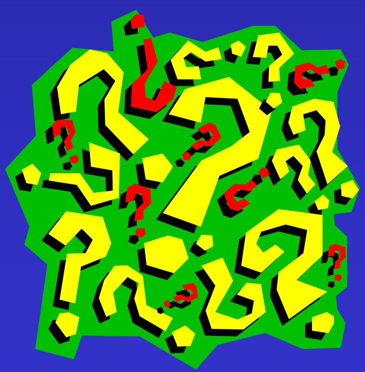
Conclusions

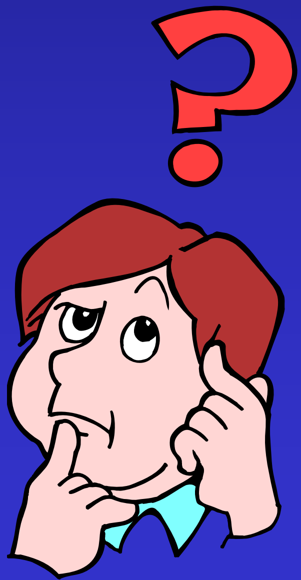
- Performance Prediction
  - Empirical hardness models can predict the run-times of randomized, incomplete, parameterized, local search algorithms
- Automated Tuning
  - We automatically find parameter settings that are better than defaults
  - Sometimes better than the best possible fixed setting
- There's no free lunch
  - Long initial training time
  - Need domain knowledge to define features for a domain (only once per domain)

The End

- Thanks to
  - Holger Hoos, Kevin Leyton-Brown, Youssef Hamadi
  - Reviewers for helpful comments
  - You for your attention ☺







Backup

Experimental setup: solvers

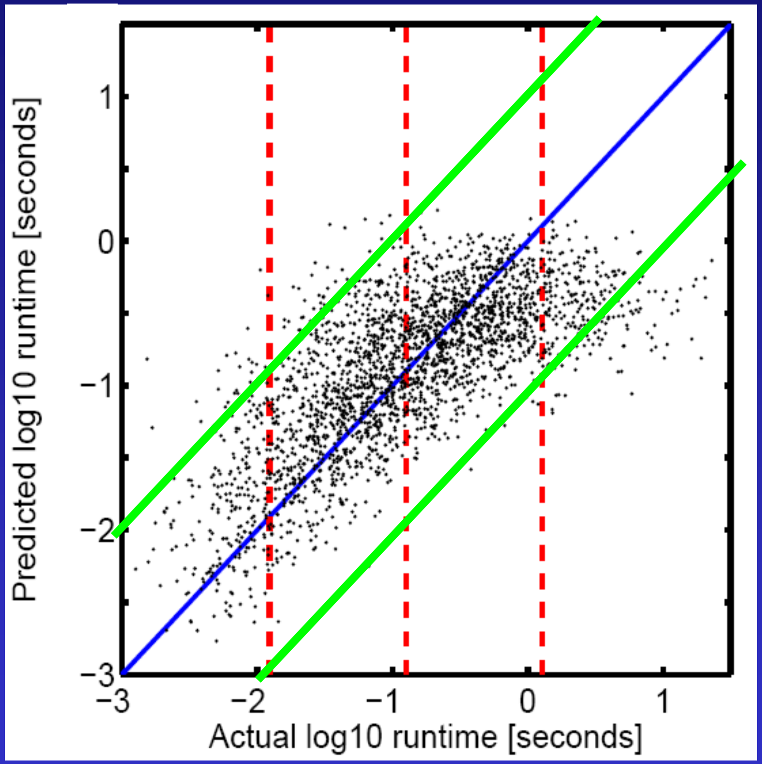
- Two SAT solvers
  - Novelty^{+} (WalkSAT variant)
    - Adaptive version won SAT04 random competition
    - Six values for noise between 0.1 and 0.6
  - SAPS (Scaling and Probabilistic Smoothing)
    - Second in above competition
    - All 30 combinations of
      - 3 values for α between 1.2 and 1.4
      - 10 values for ρ between 0 and 0.9
- Runs cut off after 15 minutes
  - Cutoff is interesting (previous talk), but orthogonal

Experimental setup: benchmarks

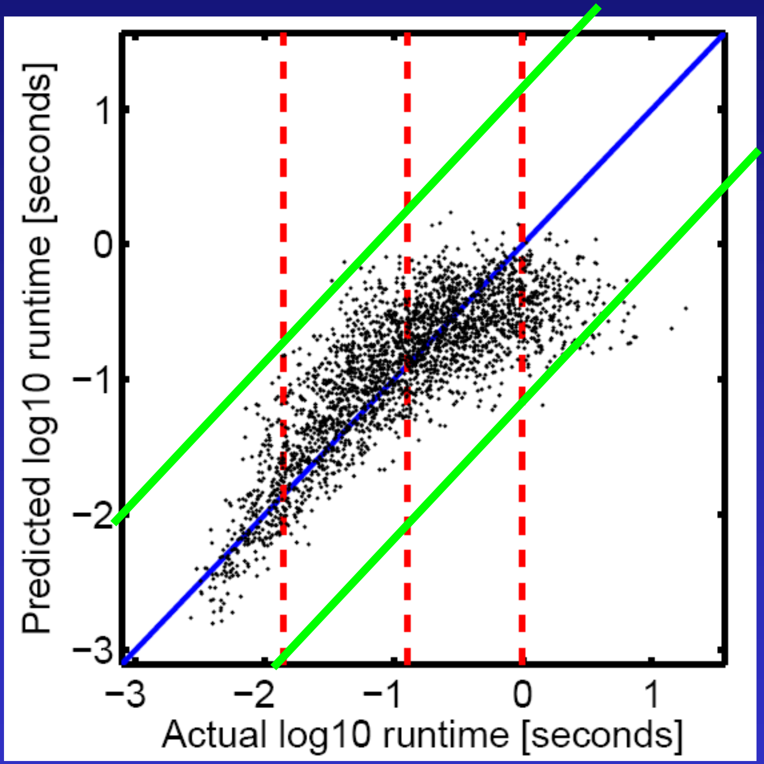
- Unstructured distributions:
  - SAT04: two generators from SAT04 competition, random
  - CV-fix: uf400 with c/v ratio 4.26
  - CV-var: uf400 with c/v ratio between 3.26 and 5.26
- Structured distributions:
  - QWH: quasi groups with holes, 25% to 75% holes
  - SW-GCP: graph colouring based on small world graphs
  - QCP: quasi group completion , 25% to 75% holes
- Mixed: union of QWH and SAT04
- All data sets split 50:25:25 for train/valid/test

Predicting median run-time

Median runtime of SAPS on CV-fix



Prediction based on single runs



Prediction based on 100 runs

Automatic tuning

- Algorithm design: new algorithm/application
  - A lot of time is spent for parameter tuning
- Algorithm analysis: comparability
  - Is algorithm A faster than algorithm B because they spent more time tuning it ? ☹
- Algorithm use in practice
  - Want to solve MY problems fast, not necessarily the ones the developers used for parameter tuning

Examples of parameters

- Tree search
  - Variable/value heuristic
  - Propagation
  - Whether and when to restart
  - How much learning
- Local search
  - Noise parameter
  - Tabu length in tabu search
  - Strength of penalty increase and decrease in DLS
  - Pertubation, acceptance criterion, etc. in ILS

Which features are most important?

- Results consistent with those for deterministic tree-search algorithms
  - Graph-based and DPLL-based features
  - Local search probes are even more important here
- Only very few features needed for good models
  - Previously observed for all-sat data [Nudelman et al. ’04]
  - A single quadratic basis function is often almost as good as the best feature subset
  - Strong correlation between features
  - Many choices yield comparable performance

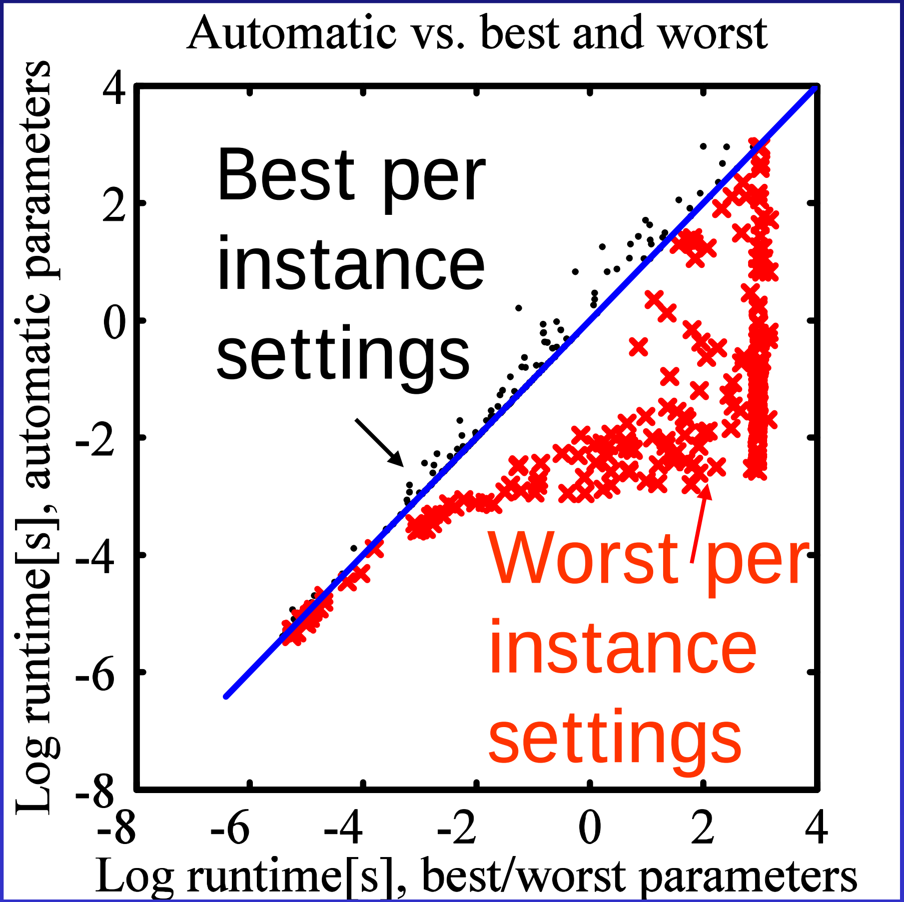
Algorithm selection based on EH models

- Given portfolio of n different algorithms $A^1,\ldots,A^n$
  - Pick best algorithm for each instance
  - E.g. satzilla
- Training:
  - Learn n separate functions
    f^j: features → runtime of algorithm j
- Test (for each new instance s_{t+1}):
  - Predict runtime $y^j_{t+1} = f^j(\phi_{t+1})$ for each algorithm
  - Choose algorithm A^j with minimal y^j_{t+1}

Experimental setup: solvers

- Two SAT solvers
  - Novelty$^+$ (WalkSAT variant)
    - Default noise setting 0.5 (=50%) for unstructured instances
    - Noise setting 0.1 used for structured instances
  - SAPS (Scaling and Probabilistic Smoothing)
    - Default setting (alpha, rho) = (1.3, 0.8)

Results for Novelty$^+$ on Mixed



Related work in automated parameter tuning

- Best default parameter setting for instance set
  - Racing algorithms [Birattari et al. ’02]
  - Local search in parameter space [Hutter ’04]
  - Fractional experimental design [Adenso-Daz & Laguna ’05]
- Best parameter setting per instance:
  algorithm selection/ algorithm configuration
  - Estimate size of DPLL tree for some algos, pick smallest [Lobjois and Lemaître, ’98]
  - Previous work in empirical hardness models [Leyton-Brown, Nudelman et al. ’02 & ’04]
  - Auto-WalkSAT [Patterson & Kautz ’02]
- Best sequence of operators / changing search strategy during the search
  - Reactive search [Battiti et al, ‘05]
  - Reinforcement learning [Lagoudakis & Littman, ’01 & ‘02]

Parameter setting based on runtime prediction

- Learn a function that predicts runtime from instance features and algorithm parameter settings (like before)
- Given a new instance
  - Compute the features (they are fix)
  - Search for the parameter setting that minimizes predicted runtime for these features

Related work: best default parameters

Find single parameter setting that minimizes expected runtime for a whole class of problems

- Generate special purpose code [Minton '93]
- Minimize estimated error [Kohavi & John '95]
- Racing algorithm [Birattari et al. '02]
- Local search [Hutter '04]
- Experimental design [Adenso-Daz & Laguna '05]
- Decision trees [Srivastava & Mediratta, '05]

Related work: per-instance selection

Examine instance, choose algorithm that will work well for it

- Estimate size of DPLL search tree for each algorithm [Lobjois and Lemaître, '98]
- [Sillito '00]
- Predict runtime for each algorithm [Leyton-Brown, Nudelman et al. '02 & '04]

Performance Prediction

- Vision: situational awareness in algorithms
  - When will the current algorithm be done ?
  - How good a solution will it find ?
- A first step: instance-aware algorithms
  - Before you start: how long will the algorithm take ?
    - Randomized $\rightarrow$ whole run-time distribution
  - For different parameter settings
    - Can pick the one with best predicted performance

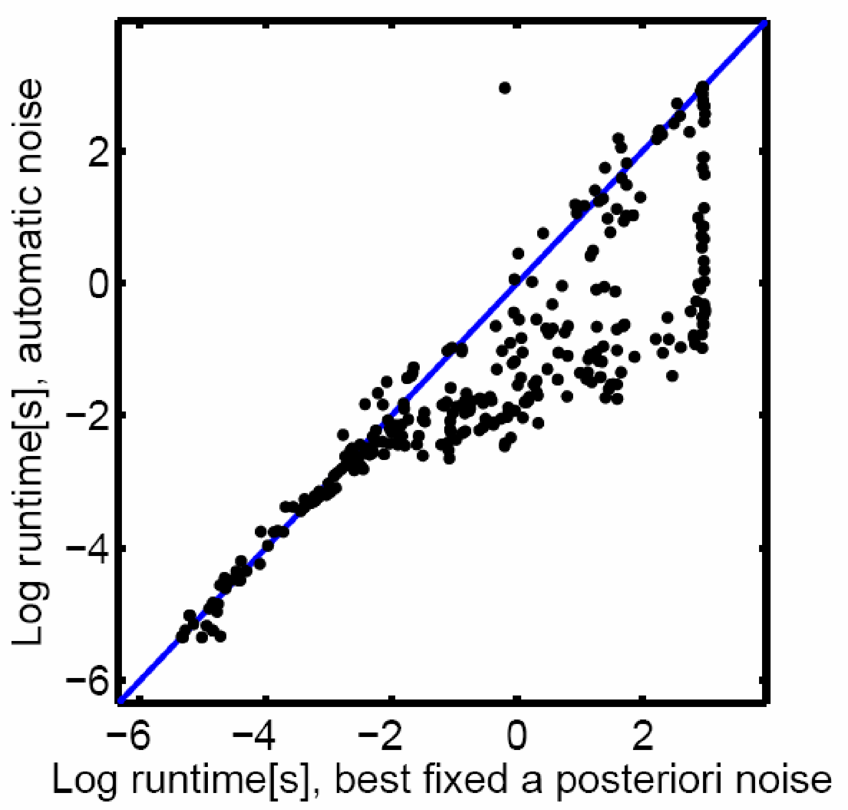
Automated parameter setting: results-old

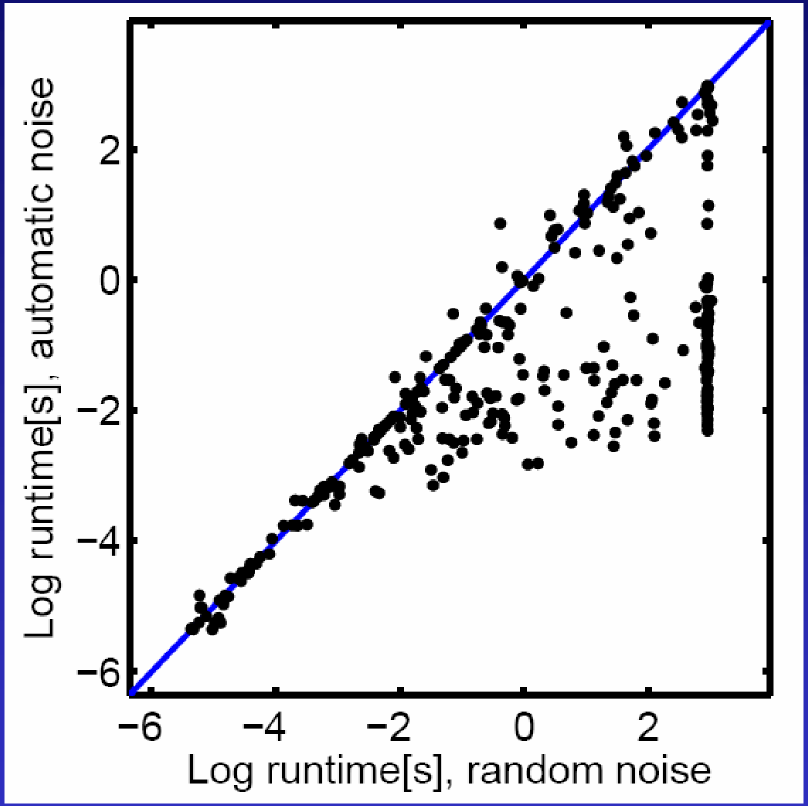
Algo	Data Set	Speedup over default params	Speedup over best fixed params for data set
Nov+	unstructured	0.89	0.89
Nov+	structured	177	0.91
Nov+	mixed	13	10.72
SAPS	unstructured	2	1.07
SAPS	structured	2	0.93
SAPS	mixed	1.91	0.93

Not the best algorithm to tune ;-)

Do you have one?

Results for Novelty^{+} on Mixed - old





Compared to best fixed parameters

Compared to random parameters