On David’s Pet Peeves*

Pet Peeve No. 10: Hand-tuned algorithm parameters.
“...many papers use different settings for different instances without explaining how those settings were derived ...”

“The rule I would apply is the following:
If different parameter settings are to be used for different instances, the adjustment process must be well-defined an algorithmic, the adjustment algorithm must be described in the paper, and the time for the adjustment must be included in all reported running times.”

---

Corollary to David’s Pet Peeves …

… paraphrased from American Scientist, July 2002

Can one replicate these experiments?

"Of course you can’t replicate my experiments. That’s the beauty of them."

Outline

• On Hypothesis Testing
• Component Reliability vs Instance Solvability
• About Instance Isomorphs
• xBed: A Testbed Environment for Experimental Algorithmics
• Call for more isomorph classes and instances …
• Case Studies (Highlights of SAT’2003 only)
  - CrossingsBed (Alenex’1999, JEA’2001)
  - SatBed (Sat’2002, Sat’2003, AMAI’2005)
  - MinOnesBed (PhD’2004, DAC’2005, …)
  - MaxOnesBed (…)
  - CspBed (PhD’2004, …)
  - BinPackingBed (…)
On Hypothesis Testing*

- A View from a Computer Scientist
- Random Instances: What's the Difference?
- A View from an Engineer
- A View from a Skeptic
- A Test of Hypothesis: An Example
- A View and Questions from a Statistician

A View from a Computer Scientist

The experiment with $k$ solvers (e.g. unitwalk below) involves a few-lines encapsulation script, e.g.

```tcl
set N 250; set M 1065
foreach RN {1215 33 1914 1066 1984 ...} {
    set Instance [Gen3Sat $N $M $RN]
    lappend Results [unitwalk $Instance]
}
puts “Statistics\n[PostProcess $Results]”
```

Statistics may appear “significant” due to large number of instances, but …

... what can we really infer once we demonstrate that these “random instances” are all VERY different?
Random Instances: What's the Difference?

orders of magnitude differences have been observed ...

See experimental results below with *unitwalk* on four 3-sat instances (N=250, M=1065) from a set of 100, repeated 128 times on replicas of each of four instances.

![Graph showing unitwalk applied to 128 sat instances from each of the four isomorphism classes.](image)

A View from an Engineer

Combinatorial solver experimental results are still published in the format below ... (for SAT and other problems)

<table>
<thead>
<tr>
<th>benchm</th>
<th>chaff</th>
<th>sato</th>
<th>satoL</th>
</tr>
</thead>
<tbody>
<tr>
<td>sch5</td>
<td>0.12</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>sch6</td>
<td>5.74</td>
<td>0.04</td>
<td>0.03</td>
</tr>
<tr>
<td>sch7</td>
<td>22.3</td>
<td>0.11</td>
<td>0.09</td>
</tr>
<tr>
<td>total</td>
<td>28.16</td>
<td>0.17</td>
<td>0.13</td>
</tr>
</tbody>
</table>

ALL instances are of different size and characteristics. Assume many more than the three shown here. Would statistician declare sato and satoL equivalent??
A View from a Skeptic

From our presentations at SAT'2002, AMAI'2005:
For each reference (eg. sch7), generate an equiv. class of at least 32 isomorphs and repeat experiment for each instance to find **intrinsic variability** of state-of-the-art SAT solvers.

<table>
<thead>
<tr>
<th>solverID</th>
<th>initV</th>
<th>minV</th>
<th>meanV</th>
<th>maxV</th>
<th>maxV/minV</th>
</tr>
</thead>
<tbody>
<tr>
<td>satoL</td>
<td>0.09</td>
<td>0.06</td>
<td>0.07</td>
<td>0.11</td>
<td>1.83</td>
</tr>
<tr>
<td>chaff</td>
<td>22.3</td>
<td>3.51</td>
<td>15.5</td>
<td>69.5</td>
<td>19.8</td>
</tr>
<tr>
<td>sato</td>
<td>0.11</td>
<td>0.09</td>
<td>238</td>
<td>t'out</td>
<td>&gt; 39,713</td>
</tr>
</tbody>
</table>

Reference instance report (from the engineer’s experiment)
A total of 9x(32+1) experiments rather than 9 shown earlier

A Test of Hypothesis: An Example

Given cumulative runtime distribution (solvability) of two solvers on the same isomorph class (of 32):

Is uw1 “better” than qt1 on this class?

Not really, 95% conf. interval for difference: (4.4 +/- 7.7) secs (t = 1.17) exp. d. 12.3/12.5 exp. d. 16.7/17.2

uf250..87-qt1
uf250..87-uw1

runtime (seconds)

solvability
A View from a Statistician

- Population: all instances within a defined class
- Sampling scheme: two-stage, with classes and instances within classes.
- Parameters of interest: mean runtimes (K solvers)
- Hypotheses: equality of means across solvers
- False positive (Type I) error rate at .05
  - multiplicity adjustments in the case of many solvers or many classes
- Sufficiently powerful experiments require
  - specification of minimum acceptable differences among solvers we hope to detect
  - specification of runtime variances

Few Questions from a Statistician

- What are the sources of variation in runtimes?
  - instances, classes, solvers
- How much variability in runtimes can be attributed to these three various factors?
- Is there evidence against null models in which
  - all class means are equal?
  - all solver means are equal?
  - solver differences constant across classes?
- Can we use block design approach here?
  Block designs form groups of plots that are similar within groups, but different across groups, then apply all treatments within each block.
Component Reliability vs Instance Solvability*

Two sides of the SAME coin

• replicated components in reliability experiments

• instances of isomorphs in solver experiments

• the same statistical methods for both

Same Coin: Reliability vs Solvability

Side 1 of the coin: Reliability of a critical component

- environment1: 30 mins, maximize lifetime
- environment2: 30 years

Side 2 of the coin: Solvability of a combinatorial problem (nug30)

- local search: 1--60 seconds (1 processor), minimize runtime
- branch-and-bound: 7 days, (2510 processors) RELIABLY!!
Example: Reliability vs Solvability

Component
(A/C unit of Boeing 720)

What lifetime?
(213 total)

Component replicas

MTTF = 93.3 hours
Std = 107 hours

(lifetime)

(acFailures, exp(-x))

Component replicas

LA-\(x*y\) problem

What runtime to optimally place \(x\) warehouses to minimize travel to \(y\) locations?
(24 total)

Problem replicas

Median = 2
Mean = 14.6
Std = 25.1

Is it still surprising that “median” is preferred (by some) to mean/standard deviation?

About Instance Isomorphs*

• introduced for learning experiments by H. Simon (1969) ... and Tower of Hanoi still being published (ACCSS’2000)

• introduced for performance evaluation of algorithms:
  ICCAD’98, IWLS’99, SSST’2001, JEA’2001, SAT2002 ...
  (in contrast to Simon, isomorph syntax is strictly invariant)

Instance generation rules for performance evaluation of algorithms depend on the problem domain:
• almost trivial for many graph problems
• needs to be “invented” for some problem domains
• may be initiated within the scope of five generic rules
Rules on Constructing Isomorphs

Define, for a problem domain:

- **R1:** syntax
- **R2:** reference instance parameters
- **R3:** instance invariants
- **R4:** parameter sampling range
- **R5:** sampling process to generate each instance

**Calculator Dexterity** Test

**Example:** find 2 roots

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>(x^2 + bx + c)</td>
</tr>
<tr>
<td>(b)</td>
<td>(a=2, b=-10, c=-100)</td>
</tr>
<tr>
<td>(c)</td>
<td>(a:1\ digit, b:2\ digits, c:3\ digits)</td>
</tr>
<tr>
<td>(1 &lt; a &lt; 10, -100 &lt; b &lt; -9)</td>
<td>(-1000 &lt; c &lt; -99)</td>
</tr>
</tbody>
</table>

**Isomorphs Example: Simple Maze Problem**

The same maze induces large variability in these two algorithms

**Reference maze**
Left-edge alg. “appears” faster

**Reference maze -- reflected**
RDLU alg. “appears” faster

**Reference maze -- rotated**
both alg. “appear” equivalent

A single reference maze can induce an equiv. class to better analyze algorithm’s behavior
Isomorphs Example: LA Problem

Similar to rotation/translation in maze, “complexity” of facility LA problems in arrangements below is the same.

Where to place 2 warehouses to minimize travel to 7 locations?

![Image of warehouse placement examples]

- Ref problem
- Ref problem rot. -15 deg
- Ref problem rot. -30 deg

Just the “rotation of x-y” coordinates induces optimization cost variability exceeding two orders of magnitude!

Isomorphs Example: Graph Problem

Similar to rotation/translation in maze, “complexity” of crossing number problem in arrangements below is the same.

- Ref instance (crossing no = 80)
- I-class instance (crossing no = 101)

Optimum placement (crossing no = 5) for all instances in this class.

Current generation of algorithms does not scale ... crossing number max/min variability can span orders of magnitude ...
Isomorphs Example: Graph (cont)

Bigraph placement performed with “dot” heuristic
(courtesy of AT&T)

Reference graph: 2123 nodes
Crossing number = 0

Equivalence class: I
Num of instances = 64

Ideally, each instance should be embeddable with 0 crossings, but ...

Our work: better performance is attainable by using equivalence class instances while engineering the algorithm.
(Alenex1999, JEA2001)

Crossing numbers reported:
Min=20, Max=1870
Mean=933.98

Isomorphs Example: SAT Problem

Similar to rotation/translation in maze, “complexity” of Boolean circuits in arrangements below is the same

reference instance

PC-class instance

... and yet, our experiments with I-, C-, P-, PC-class instances have induced orders of magnitude variability in:

- logic optimization problems (IWLS'1999)
- BDD variable ordering problems (ICCAD'98, SSST'2001)
- SAT problems (SAT'2002, see also our web-postings ...)

… similar to rotation/translation in maze, “complexity” of Boolean circuits in arrangements below is the same

reference instance

PC-class instance

... and yet, our experiments with I-, C-, P-, PC-class instances have induced orders of magnitude variability in:

- logic optimization problems (IWLS'1999)
- BDD variable ordering problems (ICCAD'98, SSST'2001)
- SAT problems (SAT'2002, see also our web-postings ...)

Reference circuit

PC-class instance
Isomorphs Example: SAT (cont)

**Solver chaff:**
129 instances from uuf250-1065 _090_PC

**Solver chaff:**
129 instances from sched06u_v00826_PC

Same solver, two very different distributions, large variance

Isomorphs Example: BinPacking Problem

Reference Instance Solution

Class Signature Invariants

Reference Instance File

p bpfi 31 8 1 7
5 2 3 2 1 4 5 4 1 2 5 3 5 2 1 5 2 5
3 1 4 5 3 1 4 1 4 5 1 3 4

Isomorph Instance File

p bpfi 31 8 1 7
6 5 2 3 7 4 1 1 3 5 2 1 4 2 2 6 1 2
4 1 4 6 1 2 1 5 1 3 1 3 7
Isomorphs Example: ListSize=99, BinSize=13

Reference
Instance

Class
Signature
Invariants

Isomorph
Instance

Isomorphs Example: BinPacking Experiments

% ../ref2bpfi 13 1011
ListSize = 100089 WeightTotal = 100089 BinTotal = 31341
% ../../xSolver_/packX < uni_000100089_0001_0013_0013_100089.bpf

100089 0.0769231 1 31765 24900 424 423.538
100089 0.0769231 1 31758 24810 417 416.923
100089 0.0769231 1 31760 24879 419 418.538
100089 0.0769231 1 31753 24837 412 411.615
100089 0.0769231 1 31752 24826 411 410.077
100089 0.0769231 1 31759 24746 418 417.308
100089 0.0769231 1 31759 24940 418 417.462
100089 0.0769231 1 31759 24746 418 417.308

Excess Bins (from known optimum of 31,341 bins)
xBed: A Testbed Environment for Experimental Algorithmics

• a schema and an architecture
  - a single MasterBed for users
  - a user-configured UserBed in desired domain
• encapsulation class scripts
• encapsulated solvers
• a userbed configuration flow example
• a userbed schema example
• a configuration file example
• a default posting of results
• web-posted experiments from SatBed

xBed : A Schema and An Architecture

<table>
<thead>
<tr>
<th>AdminPrivileges</th>
<th>UserPrivileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>MasterBed/</td>
<td>UserBed/</td>
</tr>
<tr>
<td>Docs/</td>
<td>Benchmarks/</td>
</tr>
<tr>
<td>BIN/</td>
<td>BIN/</td>
</tr>
<tr>
<td>MasterBedConfigFile</td>
<td>UserBedConfigFile</td>
</tr>
<tr>
<td>ClassScript</td>
<td>ClassScript_/</td>
</tr>
<tr>
<td>ClassScript_/</td>
<td>ProgB</td>
</tr>
<tr>
<td>ProgA</td>
<td>ProgB_</td>
</tr>
<tr>
<td>ProgA_</td>
<td>ProgBTests</td>
</tr>
<tr>
<td>ProgATests</td>
<td>Run/</td>
</tr>
<tr>
<td>xBed</td>
<td>RunConfigFile</td>
</tr>
<tr>
<td>Any number of</td>
<td>Results/</td>
</tr>
<tr>
<td>encapsulation</td>
<td>xBed can invoke any</td>
</tr>
<tr>
<td>class</td>
<td>number of (script, program) pairs</td>
</tr>
<tr>
<td>scripts and</td>
<td>via RunConfigFiles</td>
</tr>
<tr>
<td>programs</td>
<td></td>
</tr>
</tbody>
</table>

Any number of encapsulation class scripts and programs
**xBed: Encapsulated Class Scripts**

### Class Scripts:
- xGenRef
- xGenMorph
- xGenRand
- xTranslator
- xSolver
- xAnalyzer
- xVerifier
- xGenStat
- xViewer

### Encapsulated program examples:
- sched_classic, ref2bpf, ...
- morph4cnf, morph4bpf, ...
- rand2wsf, rand2bpf, ...
- wsf2cnf, cnf2lpx, ...
- chaff, cplex, packX, ...

**NOTE:**
Only a single version of a class script can exist under MasterBed/UserBed. However, another version of a program can be maintained under UserBed.

---

**xBed: Encapsulated Solvers**

### CrossingsBed:
- dot (AT&T)
- tr01--tr23

### SatBed:
- chaff
- satire
- sato
- satoL
- walksat
- unitwalk
- qingting
- dp0

### LabsBed:
- es
- kl

### MinOnesBed:
- espresso
- scherzo
- aura2
- bsolo
- umbra
- lp_solve
- cplex

### CspBed:
- sub_SAT
- wpack
- maxsat
- LB2
- qtmax

### BinPackingBed:
- packX

### MaxOnesBed:
- umbraMax
- lp_solve
- cplex
- triglav
xBED: A UserBed Configuration Flow Example

... that can be created in the user-configuration file which is then given as an argument to xBed command

Tasks:

refGen → classGen → solver1 → ... → statsGen → rptGen

Data:

xBED: A UserBed Schema Example

Here, we have a testbed archiving a large number of MinOnes problems which are either Unate or Binate.

As shown, isomorphs have already been generated from reference instances which are based on triplets in the 'Steiner' family of increasing size.
**xBed: A Configuration File Example**

The user can keep it simple

-- using commands set, lappend, and list only, or
-- unleash the full programming environment of tcl

```tcl
set BenchmarksHome ..//Bench_DemoBedIsValid
set BenchmarksHomeNode Bench_DemoBedIsValid/Unate
set ResultsHome Test_xSolverArchive/RESULTS-Tmp

set InstanceDirList \
    SmallSteiner/_ISOMORPHS/steiner_a0009_CLR \
    SmallSteiner/_ISOMORPHS/steiner_a0015_CLR \
    
foreach InstanceDir $InstanceDirList {
    set Script "xSolver umbra,bb1 $InstanceDir .lpx -NumRepeats=16 "
    set Program "umbra (FileName)"
    lappend CommandPairs [ list $Script $Program ]

    set Script "xSolver lp_solve,S2 $InstanceDir .lpx -NumRepeats=16 "
    set Program "lp_solve (FileName)"
    lappend CommandPairs [ list $Script $Program ]
}
```

**xBed: A Default Posting of Results**

The posted results inherit the hierarchy of data instances. The results directory is distinct from instance directory and the results are placed at the leaves of instance directory structure.
SatBed: Web-posted Experiments

In mission-critical applications, number of test instances should be very large.

Note variability range for the isomorph class.

Single instance reported in a traditional experiment

In mission-critical applications, number of test instances should be very large ....

Final note: not all SAT solvers return solutions that always pass verification audit!!

A Call for Isomorph Classes and Instances

Ask not what a solver can do for a set of unrelated instances ...

... ask what solver can do for set of instances in the same isomorph class!
The Strategy:
Demonstrate at least one solver that dominates all other solvers on instances of more than one isomorph class.

The Challenge:

A Case Study: Highlights from SAT’2003*

QingTing: A Fast SAT Solver Using Local Search and Efficient Unit Propagation
Xiao Yu Li Matthias F. Stallmann Franc Brglez
Motivation: UnitWalk Algorithm

Incomplete local search algorithm

Uses unit propagation intensively

Competitive with other solvers such as chaff, sato (complete solvers) and WalkSAT, GSAT (incomplete solvers)

This Talk: A Look at Two Benchmarks

Structured benchmark: sched6.cnf
Variables = 808
Clauses = 12024
PC class size = 32

Random 3-SAT benchmark: uf250-1065-087
Variables = 250
Clauses = 1065
PC class size = 128
sched6: UW1 vs Chaff (1)

Comparison between UnitWalk1 and chaff on a structured instance.

Exponential Distribution
Mean = 18.0  Std deviation = 20.1

UW1 solves the instance with less mean runtime and less variation than chaff.

sched6: UW1 vs Chaff (2)
Comparison between UnitWalk1 and chaff on a randomly generated instance.

UW1 outperforms chaff again.
Our Solver: QingTing1

How to improve UnitWalk?

• UnitWalk uses counter-based adjacency list as underlying data structure
  1. Improve its unit propagation

QingTing1 (QT1) uses two well-known unit propagation techniques used in complete solvers:

• sato's unit propagation algorithm
• chaff’s watched literal lazy data structure

sched06: QT1 vs UW1 (1)

How much better can we do with better unit propagation techniques on a structured instance?

exp. d. 0.20/0.19
sched06: QT1 vs UW1  (2)

With all other parameters, e.g. flips, approximately the same, QT1 runs five times faster than UW1.

uf250-1065-87: QT1 vs UW1  (1)

How much better can we do on a randomly generated instance?
Not much! The 5%-level t-test shows no difference. Lack of improvement due to cost of maintaining dynamic data structure.

Switching Strategy

Another Local search algorithm: WalkSAT

Observations:
1) QT1 works well on structured instances, but not on random instances
2) WalkSAT-Novelty works well on random SAT instances

Questions:
1) How do we know which one to use (switching strategy)?
2) How do we decide with low cost?
Switching Strategy: Method A (1)

Formula empty when each clause is empty or satisfied.

Switching Strategy: Method A (2)

Method A measures \# of random assignments and normalize wrt \# of variables:

\[
\text{random assignment } \% = \frac{\# \text{ of random assignments}}{\# \text{ of variables}}
\]

Repeat the process 128 times and calculate the average.
**Method A: An Example**

Random Assignment

\[(x \; y \; z) \quad \text{(-x \; y)} \quad \text{(-x \; y \; z)} \quad \text{(-y \; -z)} \quad \text{(-x \; z)}\]

\[
\begin{align*}
  x &= 1 & (y) \quad (y \; z) \quad (-y \; -z) \quad (z) \\
  z &= 0 & (y) \quad (y) \quad () \\
  y &= 0 & () \quad () \quad ()
\end{align*}
\]

formula empty!

**Method A: Experiments**

<table>
<thead>
<tr>
<th>Benchmark Name</th>
<th>Random Assignment % Mean/Std</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured</td>
<td>\begin{align*} \text{bw_large_a} &amp; \quad 0.98/0.05 \end{align*}</td>
<td>near-exp</td>
</tr>
<tr>
<td>Instances</td>
<td>\begin{align*} \text{logistics_b} &amp; \quad 0.98/0.06 \end{align*}</td>
<td>near-exp</td>
</tr>
<tr>
<td>Random Instances</td>
<td>\begin{align*} \text{uf250_1065_27} &amp; \quad 0.98/0.06 \end{align*}</td>
<td>near-exp</td>
</tr>
<tr>
<td>Random Instances</td>
<td>\begin{align*} \text{uf250_1065_87} &amp; \quad 0.96/0.07 \end{align*}</td>
<td>near-exp</td>
</tr>
</tbody>
</table>

*Method A is no good. There is no difference!*
Switching Strategy: Method B

Random Assignment

Formula

Formula Empty?

Yes

Terminate

No

Repeat the procedure 128 times and calculate average Random assignments %

Method B: On the Same Example

Random Assignment

\[(x \ y \ z) \ (-x \ y) \ (-x \ y \ z) \ (-y \ -z) \ (-x \ z)\]

\[x = 1\]

\[(y) \ (y \ z) \ (-y \ -z) \ (z)\]

Unit propagate with \(y = 1\)

\[(-z) \ (z)\]

Unit propagate with \(z = 0\)

\[(\ ) \ (\ )\]

formula empty!
**Method B: Experiments (1)**

<table>
<thead>
<tr>
<th>Benchmark Name</th>
<th>Random Assignment % Mean/Std</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bw_large_a</td>
<td>0.010/0.006</td>
<td>near-exp</td>
</tr>
<tr>
<td>Instances</td>
<td></td>
<td></td>
</tr>
<tr>
<td>logistics_b</td>
<td>0.015/0.002</td>
<td>normal</td>
</tr>
</tbody>
</table>

Random assignment % has large gap between the two groups, but is it because of 3-SAT?

**Method B: Experiments (2)**

<table>
<thead>
<tr>
<th>Benchmark Name</th>
<th>Random Assignment % Mean/Std</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bw_large_a</td>
<td>0.010/0.006</td>
<td>near-exp</td>
</tr>
<tr>
<td>Instances</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bw_large_a_3sat</td>
<td>0.046/0.024</td>
<td>near-exp</td>
</tr>
<tr>
<td>logistics_b</td>
<td>0.015/0.002</td>
<td>normal</td>
</tr>
<tr>
<td>logistics_b_3sat</td>
<td>0.049/0.066</td>
<td>near-exp</td>
</tr>
</tbody>
</table>

Random assignment % increases but the gap still exists!

What about other randomly generated 3-SAT?
# Method B: Experiments (3)

<table>
<thead>
<tr>
<th>Benchmark Name</th>
<th>Random Assignment % Mean/Std</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Structured</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bw_large_a</td>
<td>0.010/0.006</td>
<td>near-exp</td>
</tr>
<tr>
<td>bw_large_a_3sat</td>
<td>0.046/0.024</td>
<td>near-exp</td>
</tr>
<tr>
<td>logistics_b</td>
<td>0.015/0.002</td>
<td>normal</td>
</tr>
<tr>
<td>logistics_b_3sat</td>
<td>0.049/0.066</td>
<td>near-exp</td>
</tr>
<tr>
<td><strong>Random</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>uf250-1065-27</td>
<td>0.150/0.025</td>
<td>normal</td>
</tr>
<tr>
<td>uf250-1065-87</td>
<td>0.156/0.026</td>
<td>normal</td>
</tr>
<tr>
<td>hgen2-v250-s2</td>
<td>0.159/0.022</td>
<td>normal</td>
</tr>
<tr>
<td>hgen2-v250-s500</td>
<td>0.161/0.019</td>
<td>near-normal</td>
</tr>
</tbody>
</table>

## QingTing2 and UnitWalk2

### QingTing2
- Switching strategy (low overhead)
- On randomly generated instances, QT2 behaves like WalkSAT-Novelt
- On structured instances, QT2 behaves like QT1

### UnitWalk2
- UnitWalk2 is the latest version of UnitWalk and it also does some structure detection
What will the improvements be on a structured instance?

QT1 runs five times faster than UW1.
sched6: QT2/QT1 vs UW2/UW1  (3)

QT1 runs five times faster than UW1

UW2 and UW1 are the same (t-test: t = 1.44 <1.97 )

sched6: QT2/QT1 vs UW2/UW1  (4)

QT1 runs five times faster than UW1

UW2 and UW1 are the same (t-test: t = 1.44 <1.97 )

QT1 and QT2 are the same (t-test: t = 1.18 <1.97 )
What about random 3-SAT instances?

UW1 performs the same as QT1 (t-test: $t = 1.88 > 1.97$)
uf250..87: QT2/QT1 vs UW2/UW1 (3)

UW1 performs the same as QT1 (t-test: $t = 1.88 > 1.97$)
UW2 outperforms UW1 by a factor of 31 …

uf250..87: QT2/QT1 vs UW2/UW1 (4)

UW1 performs the same as QT1 (t-test: $t = 1.88 > 1.97$)
UW2 outperforms UW1 by a factor of 31 …
QT2 outperforms UW2 slightly (t-test: $t = 2.24 > 1.97$)
Summary and Conclusions (1)

- QingTing1
  - Improving UnitWalk
  - Faster Unit Propagation

- QingTing2
  - Switching strategy
  - Combine QingTing1 with WalkSAT

Summary and Conclusions (2)

Future research directions

1. Use switching strategy in a complete solver

2. Use learning techniques in local search algorithms

3. Combine the theory and the experimental methodology