Dynamic Strategy Priority

Michael Rawson, Giles Reger

University of Manchester, UK

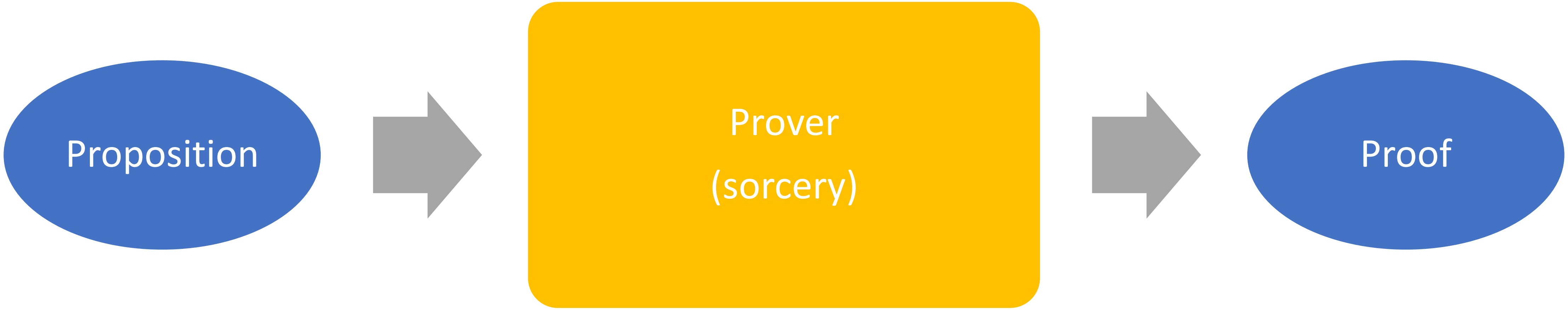
Background

Theorem Provers

Vampire

Algorithms

Automated Theorem Provers



ATP Design & Folklore

- Hard problem
- Complementary heuristics
- Best results require careful engineering
- Algorithm either solves a problem quickly, or not at all
- Small changes cause huge differences.
- Performance matters
- Generally care about *finding* proofs, not time-to-proof

Vampire

- ATP
- First-order logic with equality (+ extensions)
- Experimental testbed for this project.

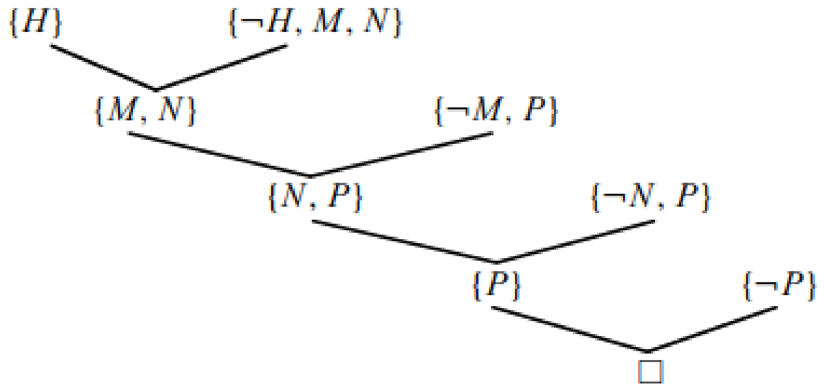
Vampire (advertisement)

- High-performance
- CASC veteran
- State-of-the-art proof technology
- Well-engineered
- Decades of experience
- Large database of problems and proofs
- http://vprover.github.io
- *No longer hard to obtain!*

Get Vampire!

Algorithm

- Pre-process (“clausify”) your problem, *negated*
- Obtain set of *clauses*
- *Resolve* clauses according to rules
- Iterate
- Produce the empty clause
- QED



Strategies

What are they?

Why does Vampire need them?

Strategies in Vampire

- Algorithm
- Parameters
- Time to run

dis+1011_24_cond=fast:drc=off:nwc=10:nicw=on:ptb=off:ssec=off:spl=sat_14

Strategies in Vampire

- Run several short strategies
- Better than one long strategy
- Execute strategy *schedules*
- Find schedules experimentally
- Problem solved?

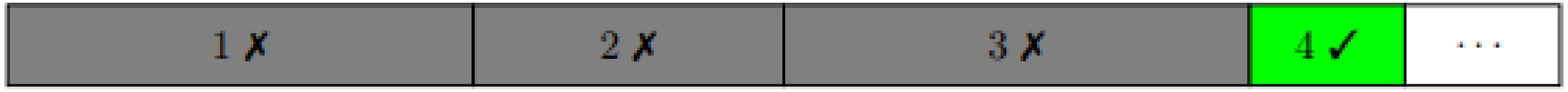
Strategy	Success?
1	No
2	No
3	Yes!
4	…

Strategy schedules are great!

- Much better than using just one strategy
- Minimal overhead
- Engineering opportunity
- Parallelism

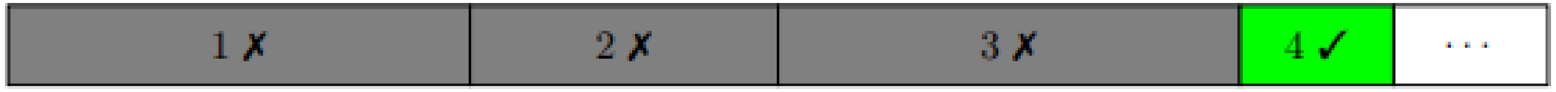
Unfortunately…

- Computationally difficult to find “best” strategy schedules
- Engineering: `schedules.cpp` unreasonable
- Brittle and rigid
- *Frequently slow due to ordering*

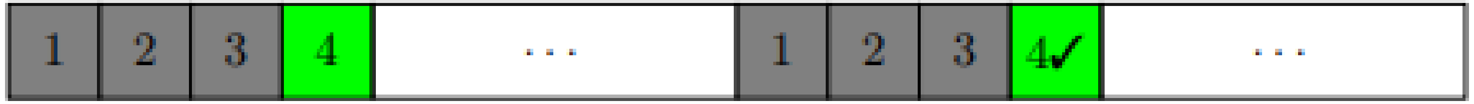


Possible fix: round-robin scheduling

- Turns this



- Into this



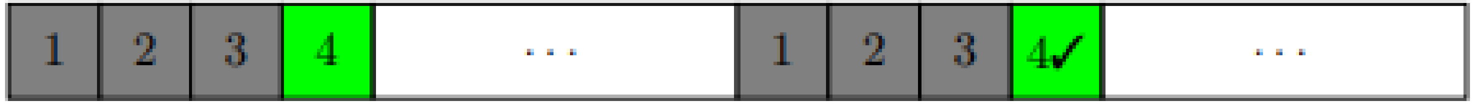
- Doesn’t fix other issues, context-switching overheads
- Not particularly “smart”

Possible fix: “static” priority

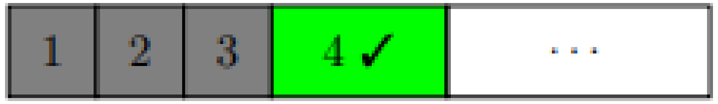
- Try and guess which strategy will work first
- Based on features of the input problem
- Vampire attempts some version of this when selecting schedules
- Little runtime overhead
- Overly ambitious?

Proposal: “dynamic” strategy priority

- Run a standard or existing schedule, but…
- Regularly stop and check running strategies
- Look for strategies which look like they’re succeeding or failing (how?)
- Prioritise succeeding strategies
- Turn this:



- Into this:



Machine Learning

Data collection & preparation

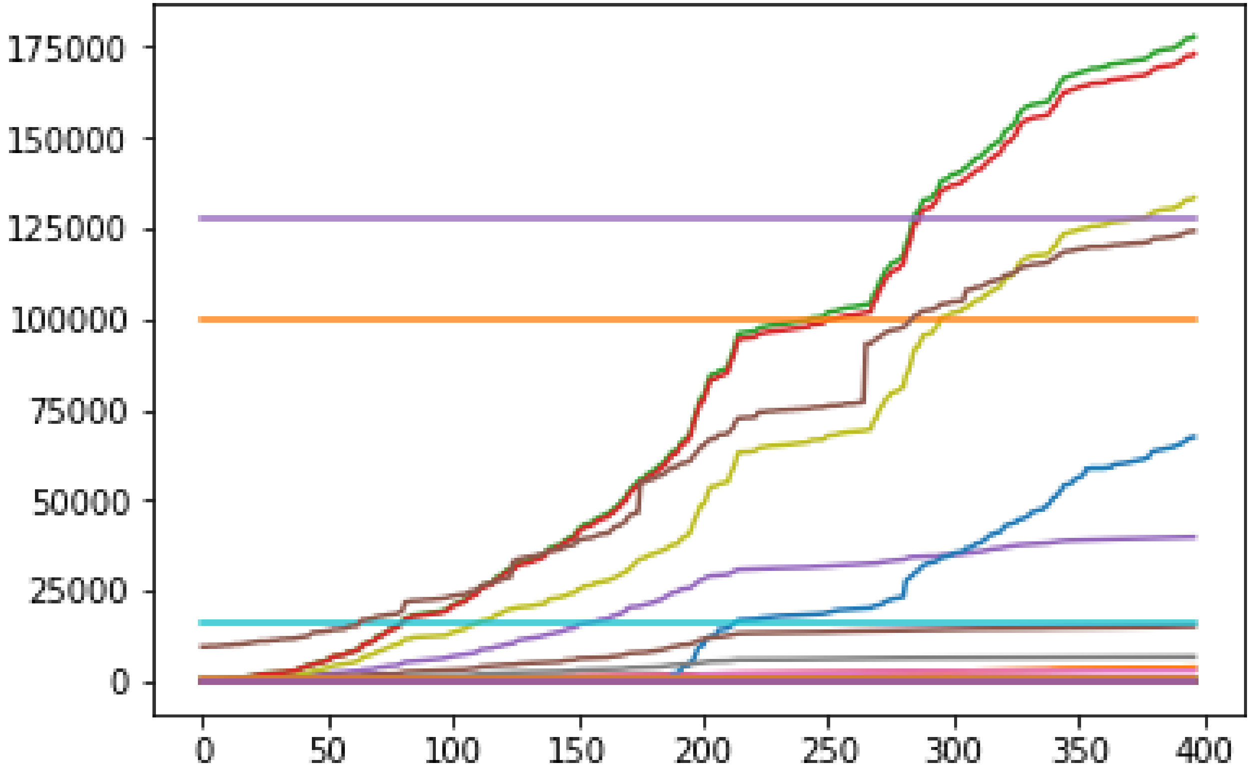
Classifier training

Evaluation

Data collection

- We want to find “something” to identify strategies
- Not sure what
- Available features: properties, statistics, options
- Report ALL the features (376 of them)
- Report a real-valued feature vector every *k* resolution steps
- Decently-large dataset of execution *traces*
- Tag traces with success or failure

Example trace



Data ideals

- Real values
- Zero mean, unit variance
- Fixed input size
- Not too voluminous
- Preferably matrix-like
- Balanced dataset

Problems and “Solutions”

- Problem: far more failing than successful traces
  - “Solution”: randomly select enough failures to match successes
- Problem: traces are of wildly differing lengths
  - “Solution”: bucket-averaging to the same size
- Problem: want to predict based on *running* (not complete) strategies
  - “Solution”: chop up traces to yield several traces at different stages
- Problem: traces contain high-variance data
  - Solution: feature scaling!

Example trace (after pre-processing)

Classifiers

- Several off-the-shelf technologies:
  - “Vanilla” neural network
  - Convolutional neural network
  - Gated Recurrent Unit (a recurrent neural network)
- All neural networks, 1 hidden layer. Other suggestions welcome!
- Thanks to *Keras* and *Scikit-learn*.

Results (5-fold cross-validation)

Method	Mean accuracy (%)	Standard deviation (%)
Simple NN	81.5	2.0
Convolutional NN	82.4	3.1
Recurrent NN	83.9	1.9

Results

- High accuracy (?!)
- Surprisingly little difference between NN methods
- Reasonably consistent across a dataset of around 10,000 traces
- Little tweaking required

Selection

- Despite expectation: simple NN suffices
- Much more performant than other classifiers
- Simple to implement!

Implementation

Integrating classifier

Scheduling modifications

Benchmark

Integrating classifiers

- Challenging for more-sophisticated classifiers
- Not too bad for the simple classifier chosen
- Train network in Python
- Generate C(++) code
- No dependencies

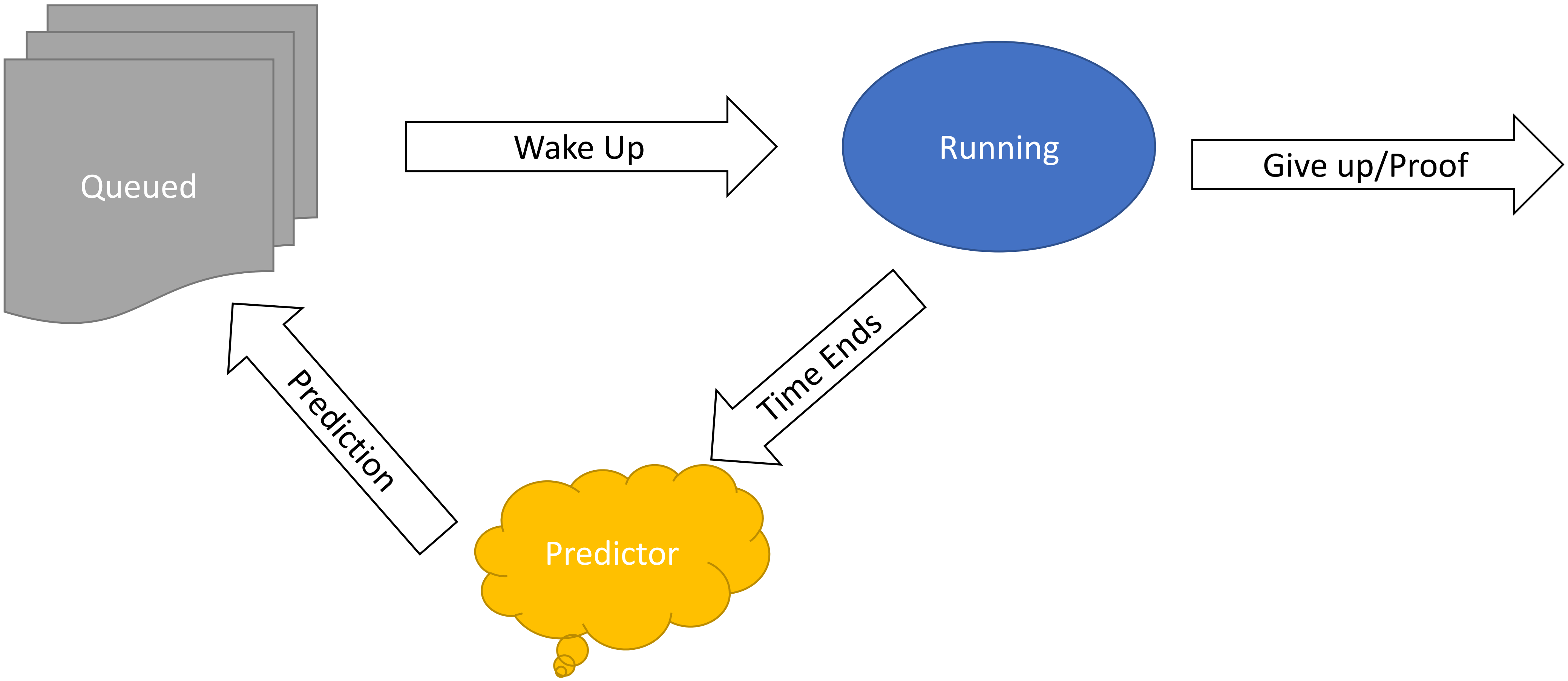
Scheduling issues

- Ideally, run all strategies simultaneously. But…
  - Memory pressure
  - Start-up time
  - Less good for the common case: short strategies
- If we don’t have all strategies running at once, what’s the point?

Compromise

- Pool of running processes
- Queue of paused (or non-started) processes
- When the pool needs filling, take processes from queue
- Regularly remove processes from the run-pool and re-evaluate them
- Still partially sequential, avoids some issues
- Only has all strategies running simultaneously in a corner case

System overview



Benchmarks

- Thanks to StarExec, TPTP
- Compared to baseline without modification on around 20k problems
- Not good for solving theorems: 585 vs 111 distinct proofs found
- **Better for speed: 10.5s/proof vs 8.7s/proof!**
- Also compared to round-robin: better in both aspects

Wrap Up

Conclusions

- Very early work
- Classifier performance is pleasantly surprising
- Implementation issues
- Perhaps an unfair comparison with baseline Vampire?
- Plenty of room for improvement and future exploration

Questions