Measuring progress to predict success: Can a good proof strategy be evolved?

Giles Reger[1], Martin Suda[2]

[1]School of Computer Science, University of Manchester, UK
[2]TU Wien, Vienna, Austria

AITP 2017 – Obergurgl, March 29, 2017

Vampire advertising

Vampire

- a “reasonably well-performing” first-order ATP
- unfortunately not open source
- known to be notoriously hard to obtain

Vampire advertising

Vampire

- a “reasonably well-performing” first-order ATP
- unfortunately not open source
- known to be notoriously hard to obtain

Things are actually not so dark:

- email me, I can send you an executable
- find one at `https://www.starexec.org/`
- (don't) look for the source at:
  `http://www.cs.miami.edu/~tptp/CASC/J8/Entrants.html`

Outline

1. The role of strategies in modern ATPs
2. Proving with orderings
3. How to evolve a precedence?
4. Conclusion

The role of strategies in modern ATPs

Strategy:

- there are many-many options to setup the proving process
- a strategy is a concrete way to do this setup

The role of strategies in modern ATPs

Strategy:

- there are many-many options to setup the proving process
- a strategy is a concrete way to do this setup

From the ATP lore

If a strategy solves a problem then it typically solves it within a short amount of time (say, 5 seconds).

The role of strategies in modern ATPs

Strategy:

- there are many-many options to setup the proving process
- a strategy is a concrete way to do this setup

From the ATP lore

If a strategy solves a problem then it typically solves it within a short amount of time (say, 5 seconds).

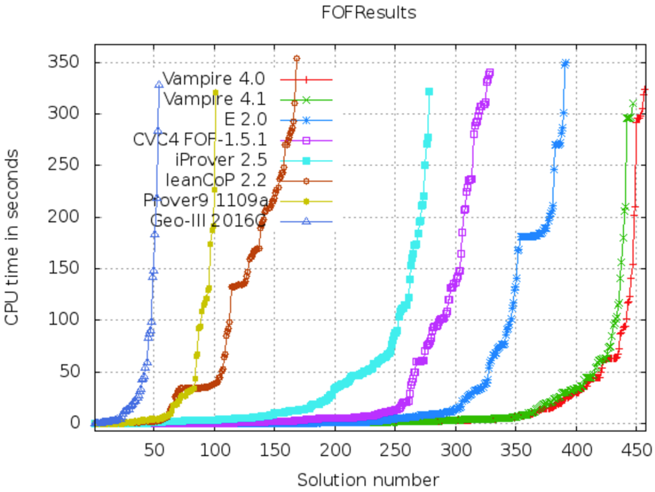
What does this mean?

- There is no single best strategy
- It's usually better to start something else than to wait
- Strategy Scheduling (portfolio approach)

CASC-mode: a conditional schedule of strategies

```
case Property::FNE:
  if (atoms > 2000) {
    quick.push("dis+1011_40_bs=on:cond=on:gs=on:gsaa=from_current:nwc=1:sfr=on:ssfp=1000:ssfq=2.0:smm=sco
    quick.push("lrs+1011_3_nwc=1:stl=90:sos=on:spl=off:sp=reverse_arity_133");
    quick.push("dis-10_5_cond=fast:gsp=input_only:gs=on:gsem=off:nwc=1:sas=minisat:sos=all:spl=off:sp=occu
    quick.push("lrs+1011_5_cond=fast:gs=on:nwc=2.5:stl=30:sd=3:ss=axioms:sdd=off:sfr=on:ssfp=100000:ssfq=
    quick.push("lrs-3_5:4_bs=on:bsr=on:cond=on:fsr=off:gsp=input_only:gs=on:gsaa=from_current:gsem=on:lcm=
  }
  else if (atoms > 1200) {
    quick.push("lrs+1011_5_cond=fast:gs=on:nwc=2.5:stl=30:sd=3:ss=axioms:sdd=off:sfr=on:ssfp=100000:ssfq=
    quick.push("dis+1011_8_bsr=unit_only:cond=fast:fsr=off:gs=on:gsaa=full_model:nm=0:nwc=1:sas=minisat:so
    quick.push("dis+11_7_gs=on:gsaa=full_model:lcm=predicate:nwc=1.1:sas=minisat:ssac=none:ssfp=1000:ssfq=
    quick.push("ins+11_5_br=off:gs=on:gsem=off:igbrr=0.9:igrr=1/64:igrp=1400:igrpq=1.1:igs=1003:igwr=on:lc
  }
  else {
    quick.push("dis+11_7_16");
    quick.push("dis+1011_5:4_gs=on:gsssp=full:nwc=1.5:sas=minisat:ssac=none:sdd=off:sfr=on:ssfp=40000:ssfq
    quick.push("dis+1011_40_bs=on:cond=on:gs=on:gsaa=from_current:nwc=1:sfr=on:ssfp=1000:ssfq=2.0:smm=sco
    ...
```

Results for FOF division of CASC 2016[1]



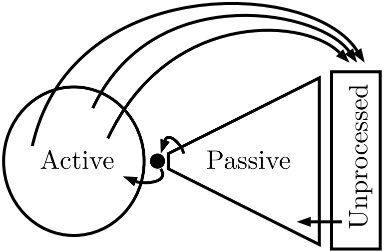
[1] www.cs.miami.edu/~tptp/CASC/J8/WWWFiles/ResultsPlots.html

Outline

1. The role of strategies in modern ATPs
2. Proving with orderings
3. How to evolve a precedence?
4. Conclusion

The Saturation Loop

Saturate a set of clauses with respect to an inference system



- Initially: the input clauses start in passive, active is empty
- Given clause: selected from passive as the next to be processed
- Move the give clause from active to passive and perform all inferences between clauses in active and the given clause

Resolution

$$\frac{A \vee C_1 \quad \neg A' \vee C_2}{(C_1 \vee C_2)\theta},$$

Factoring

$$\frac{A \vee A' \vee C}{(A \vee C)\theta},$$

where, for both inferences, $\theta = \mathrm{mgu}(A, A')$ *and* A *is not an equality literal, and* A *and* $\neg A'$ *are (strictly) maximal in their respective clauses*

Superposition

$$\frac{l \simeq r \vee C_1 \quad L[s]_p \vee C_2}{(L[r]_p \vee C_1 \vee C_2)\theta} \quad \text{or} \quad \frac{l \simeq r \vee C_1 \quad t[s]_p \otimes t' \vee C_2}{(t[r]_p \otimes t' \vee C_1 \vee C_2)\theta},$$

where $\theta = \mathrm{mgu}(l, s)$ *and* $r\theta \not\succeq l\theta$ *and, for the left rule* $L[s]$ *is not an equality literal, and for the right rule* $\otimes$ *stands either for* $\simeq$ *or* $\not\simeq$ *and* $t'\theta \not\succeq t[s]\theta$

EqualityResolution

$$\frac{s \not\simeq t \vee C}{C\theta},$$

where $\theta = \mathrm{mgu}(s, t)$

EqualityFactoring

$$\frac{s \simeq t \vee s' \simeq t' \vee C}{(t \not\simeq t' \vee s' \simeq t' \vee C)\theta},$$

where $\theta = \mathrm{mgu}(s, s')$, $t\theta \not\succeq s\theta$, *and* $t'\theta \not\succeq s'\theta$

How important could an ordering be?

Consider proving a formula

$$\psi = \bigwedge_{i=1,\ldots,n} (a_i \vee b_i) \to \bigwedge_{i=1,\ldots,n} (a_i \vee b_i)$$

How important could an ordering be?

Consider proving a formula

$$\psi = \bigwedge_{i=1,\ldots,n} (a_i \vee b_i) \to \bigwedge_{i=1,\ldots,n} (a_i \vee b_i)$$

- a naive clausification of $\neg\psi$ has $2^n + n$ clauses!

How important could an ordering be?

Consider proving a formula

$$\psi = \bigwedge_{i=1,\ldots,n} (a_i \vee b_i) \rightarrow \bigwedge_{i=1,\ldots,n} (a_i \vee b_i)$$

- a naive clausification of $\neg\psi$ has $2^n + n$ clauses!
- goes down to $3n + 1$ with *Tseitin encoding:*

$$(a_i \vee b_i), \quad (\neg m_i \vee \neg a_i), (\neg m_i \vee \neg b_i), \quad (m_1 \vee \ldots \vee m_n),$$

where m_i is a name for $\neg a_i \wedge \neg b_i$

How important could an ordering be?

Consider proving a formula

$$\psi = \bigwedge_{i=1,\dots,n} (a_i \vee b_i) \rightarrow \bigwedge_{i=1,\dots,n} (a_i \vee b_i)$$

- a naive clausification of $\neg\psi$ has $2^n + n$ clauses!
- goes down to $3n + 1$ with *Tseitin encoding:*

$$(a_i \vee b_i), \quad (\neg m_i \vee \neg a_i), (\neg m_i \vee \neg b_i), \quad (m_1 \vee \ldots \vee m_n),$$

where m_i is a name for $\neg a_i \wedge \neg b_i$

Question:

What will superposition derive under an ordering where

$$m_i \succ a_j \text{ and } m_i \succ b_j \text{ for every } i \text{ and } j?$$

Choosing an ordering

Orderings typically used in ATPs:

- Knuth-Bendix Ordering (KBO),
- Lexicographic Path Ordering (LPO)

Choosing an ordering

Orderings typically used in ATPs:

- Knuth-Bendix Ordering (KBO),
- Lexicographic Path Ordering (LPO)

Both determined by a precedence on the problem's signature:

- a linear order on the symbols occurring in the problem

We have $n!$ possibilities for choosing the ordering

Choosing an ordering

Orderings typically used in ATPs:

- Knuth-Bendix Ordering (KBO),
- Lexicographic Path Ordering (LPO)

Both determined by a precedence on the problem's signature:

- a linear order on the symbols occurring in the problem

We have $n!$ possibilities for choosing the ordering

ATPs typically provide a few schemes for fixing the precedence

Example

- Vampire: arity, reverse arity, occurrence
- E: frequency (`invfreq`), many more

Playing with precedence

Rules of the game

- Fix a single theorem proving strategy in Vampire:
  `-av off -sa discount -awr 10 -lcm predicate`
- Then by varying only the precedence
- try to solve as many TPTP problems as possible

Playing with precedence

Rules of the game

- Fix a single theorem proving strategy in Vampire:
  `-av off -sa discount -awr 10 -lcm predicate`
- Then by varying only the precedence
- try to solve as many TPTP problems as possible

TPTP library, version 6.4.0, contains 17280 first-order problems

Playing with precedence

Rules of the game

- Fix a single theorem proving strategy in Vampire:
  `-av off -sa discount -awr 10 -lcm predicate`
- Then by varying only the precedence
- try to solve as many TPTP problems as possible

TPTP library, version 6.4.0, contains 17280 first-order problems

- 9277 solved by "arity" in 300s

Playing with precedence

Rules of the game

- Fix a single theorem proving strategy in Vampire:
  `-av off -sa discount -awr 10 -lcm predicate`
- Then by varying only the precedence
- try to solve as many TPTP problems as possible

TPTP library, version 6.4.0, contains 17280 first-order problems

- 9277 solved by "arity" in 300s
- 9457 solved by "frequency" in 300s (Thank you, Stephan!)

Playing with precedence

Rules of the game

- Fix a single theorem proving strategy in Vampire:
  `-av off -sa discount -awr 10 -lcm predicate`
- Then by varying only the precedence
- try to solve as many TPTP problems as possible

TPTP library, version 6.4.0, contains 17280 first-order problems

- 9277 solved by "arity" in 300s
- 9457 solved by "frequency" in 300s (Thank you, Stephan!)
- ∼12500 solved in 300s by either `casc` or `casc_sat` mode

How good is a random precedence?

From the previous page:

- 9277 by “arity” in 300s
- 9457 by “frequency” in 300s

How good is a random precedence?

From the previous page:

- 9277 by "arity" in 300s
- 9457 by "frequency" in 300s

Shuffle once:

- $\sim$7100 solved with a random precedence (3s)
- $\sim$8450 solved with a random precedence (60s)
- $\sim$9100 solved with a random precedence (300s)

How good is a random precedence?

From the previous page:

- 9277 by "arity" in 300s
- 9457 by "frequency" in 300s

Shuffle once:

- $\sim$7100 solved with a random precedence (3s)
- $\sim$8450 solved with a random precedence (60s)
- $\sim$9100 solved with a random precedence (300s)

Shuffle a few times:

- 9387 solved in a union of 9 independent random precedence 60s runs (1678 problems in the grey zone)

Scheduler with Dice and Harmonic numbers

Quesion:

If the only way to vary a strategy would be to randomise the precedence, how many TPTP problems could I solve given a time limit of 300s per problem?

Scheduler with Dice and Harmonic numbers

Quesion:

If the only way to vary a strategy would be to randomise the precedence, how many TPTP problems could I solve given a time limit of 300s per problem?

The setup:

- for $i = 1$ to 100
  run over TPTP with a $seed = i$ and time limit $300.0/i$ s
- $17280 \cdot H_{100} \cdot 300s \approx 311$ days of computation

Scheduler with Dice and Harmonic numbers

Quesion:

If the only way to vary a strategy would be to randomise the precedence, how many TPTP problems could I solve given a time limit of 300s per problem?

The setup:

- for $i = 1$ to 100
  run over TPTP with a $seed = i$ and time limit $300.0/i$ s
- $17280 \cdot H_{100} \cdot 300s \approx 311$ days of computation

How many slices could a reasonably good schedule use?

Scheduler with Dice and Harmonic numbers

Quesion:

If the only way to vary a strategy would be to randomise the precedence, how many TPTP problems could I solve given a time limit of 300s per problem?

The setup:

- for $i = 1$ to 100
  run over TPTP with a $seed = i$ and time limit $300.0/i$ s
- $17280 \cdot H_{100} \cdot 300s \approx 311$ days of computation

How many slices could a reasonably good schedule use?

3.0s (7093) 3.0s (330) 3.1s (192) 3.2s (111) 3.3s (101) 4.4s (163) 4.5s (87) 4.8s (79) 5.0s (64) 6.2s (108) 9.6s (156) 11.1s (104) 11.5s (64) 21.4s (169) 205.3s (736)

Solves 9557 problems (9566 on validation set)

Outline

1. The role of strategies in modern ATPs
2. Proving with orderings
3. How to evolve a precedence?
4. Conclusion

The Slowly-Growing-Search-Space heuristic

SGSS in a nutshell:

A strategy that leads to a slowly growing search space will likely be more successful at finding a proof (in reasonable time) than a strategy that leads to a rapidly growing one.

The Slowly-Growing-Search-Space heuristic

SGSS in a nutshell:

A strategy that leads to a slowly growing search space will likely be more successful at finding a proof (in reasonable time) than a strategy that leads to a rapidly growing one.

Intuition:

- Can we find the proof before it chokes?
- Since it's hard to predict if we are getting close ...
- ... try to postpone the choking until we (hopefully) get there.

The Slowly-Growing-Search-Space heuristic

SGSS in a nutshell:

A strategy that leads to a slowly growing search space will likely be more successful at finding a proof (in reasonable time) than a strategy that leads to a rapidly growing one.

Intuition:

- Can we find the proof before it chokes?
- Since it's hard to predict if we are getting close ...
- ... try to postpone the choking until we (hopefully) get there.

Successfully applied in previous work on literal selection [RSV16]

Using SGSS to look for a good precedence

Main complication:

- Ordering must be fixed during the entire proof attempt

Using SGSS to look for a good precedence

Main complication:

- Ordering must be fixed during the entire proof attempt

Idea

Look for strategies which minimize the number of derived clauses after a certain (small) number of iterations of the saturation loop.

Using SGSS to look for a good precedence

Main complication:

- Ordering must be fixed during the entire proof attempt

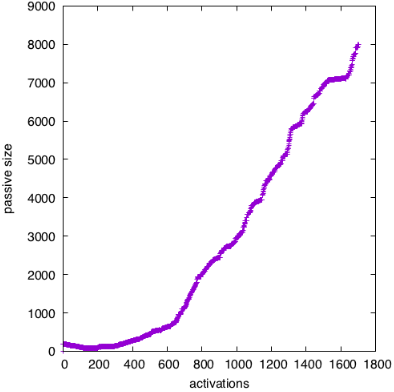
Idea

Look for strategies which minimize the number of derived clauses after a certain (small) number of iterations of the saturation loop.

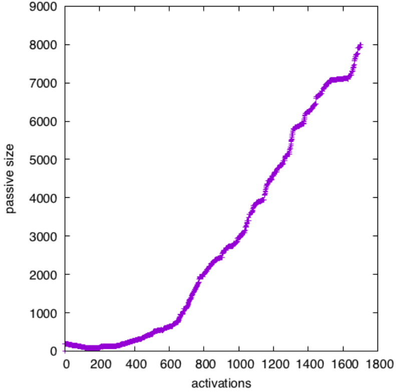
Can this work in practice?

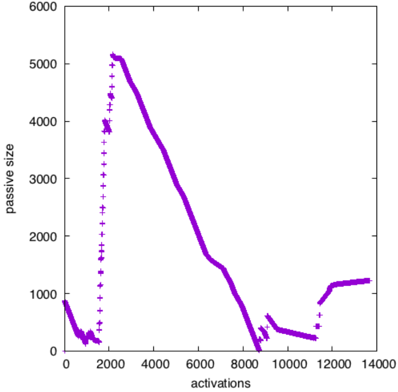
- Probably not under tight time constraints.
- In any case:
  Are there actually any good precedences out there?
- Possible application:
  solve hard previously unsolved problems

A(n a)typical development of the passive set's size



A(n a)typical development of the passive set's size





Can it possibly work?

Using the 9 independent random-precedence 60 second runs

On the set P of 1678 problems from the "grey zone"

Record size of passive every 100 activations

Compute nine respective sums s_i until the first stream stops:
$S_1(p) = s_1(p, 0) + s_1(p, 100) + s_1(p, 200) + \ldots$
$\ldots$
$S_9(p) = s_9(p, 0) + s_9(p, 100) + s_1(p, 200) + \ldots$

Denote the average $S_i(p)$ over (un)successful runs i as $\bar{S}_{(un)succ}(p)$

For how many $p \in P$ is $\bar{S}_{succ}(p) < \bar{S}_{unsucc}(p)$?

Can it possibly work?

Using the 9 independent random-precedence 60 second runs

On the set P of 1678 problems from the "grey zone"

Record size of passive every 100 activations

Compute nine respective sums s_i until the first stream stops:
$S_1(p) = s_1(p, 0) + s_1(p, 100) + s_1(p, 200) + \ldots$
$\ldots$
$S_9(p) = s_9(p, 0) + s_9(p, 100) + s_1(p, 200) + \ldots$

Denote the average $S_i(p)$ over (un)successful runs i as $\bar{S}_{(un)succ}(p)$

For how many $p \in P$ is $\bar{S}_{succ}(p) < \bar{S}_{unsucc}(p)$?

Answer: 1130 (out of 1669)

How did we evolve, then?

How did we evolve, then?

Optimize_precedence(p, t_1, t_2)

- run "frequency" for 1s to establish *act_cnt*
- spawn a population Π of n random precedences
- the fitness of $\pi \in \Pi$ is $S_\pi(p)$:
  the sum of the passive set sizes during a run on p
  summing every step from 0 to *act_cnt* activations
- loop for t_1 seconds:
  - pick a $\pi \in \Pi$
  - randomly (adaptively) perturb π to obtain π'
  - evaluate π' as above
  - keep the better of π and π'
- Finally, run with π_{best} for t_2 seconds

Results

First a test run:

- optimizing for 300s and final run for 60s: 8965
- “control” where the final run is “frequency”: 8888

Results

First a test run:

- optimizing for 300s and final run for 60s: 8965
- "control" where the final run is "frequency": 8888

The "long" run:

- 1200s optimizing, 300s final run: 9604
- solved a rating 1.0 problem: `SWV978-1`

Results

First a test run:

- optimizing for 300s and final run for 60s: 8965
- "control" where the final run is "frequency": 8888

The "long" run:

- 1200s optimizing, 300s final run: 9604
- solved a rating 1.0 problem: `SWV978-1`

How many have solved in total?

- "frequency" 300s: 9457 (40 uniques)
- all the "harmonic" runs: 10030 (202 uniques)
- the long optimizing run: 9604 (87 uniques)
- In total: 10176

Conclusion

Lessons learned:

- A good ordering can make a difference
- If out of ideas, check out what E does
- The slowly-growing-search-space heuristic works!

Conclusion

Lessons learned:

- A good ordering can make a difference
- If out of ideas, check out what E does
- The slowly-growing-search-space heuristic works!

Future work:

- Where else could SGSS be applied?
- How to make it more useful in a time-critical setting?

Conclusion

Lessons learned:

- A good ordering can make a difference
- If out of ideas, check out what E does
- The slowly-growing-search-space heuristic works!

Future work:

- Where else could SGSS be applied?
- How to make it more useful in a time-critical setting?

Thank you for your attention!