Tactic Learning for Coq

Lasse Blaauwbroek, Josef Urban

Czech Institute for Informatics, Robotics and Cybernetics
Czech Technical University in Prague

April 8, 2019

Tactic-level automation:

Idea: Learn from human-written proof scripts

Try to match proof states with the right tactic

Advantage: We can make use of custom made, domain specific tactics written by clever humans

System Components

System Components

- ▷ Proof Recording

System Components

- Proof Recording
- Tactic Prediction

System Components

- Proof Recording
- Tactic Prediction
- Proof Search

System Components

- ▷ Proof Recording
- ▷ Tactic Prediction
- ▷ Proof Search
- ▷ Proof Reconstruction

Design goals

Design goals

- ▷ User Friendly
- ▷ Installation Friendly
- ▷ Integration Friendly
- ▷ Maintenance Friendly

Design goals

- ▷ User Friendly   Online learning, minimal configuration, works everywhere
- ▷ Installation Friendly
- ▷ Integration Friendly
- ▷ Maintenance Friendly

Design goals

- User Friendly — Online learning, minimal configuration, works everywhere
- Installation Friendly — No external dependencies
- Integration Friendly
- Maintenance Friendly

Design goals

- ▷ User Friendly — Online learning, minimal configuration, works everywhere
- ▷ Installation Friendly — No external dependencies
- ▷ Integration Friendly — Coq plugin, with minor modifications to Coq
- ▷ Maintenance Friendly

Design goals

- ▷ User Friendly — Online learning, minimal configuration, works everywhere
- ▷ Installation Friendly — No external dependencies
- ▷ Integration Friendly — Coq plugin, with minor modifications to Coq
- ▷ Maintenance Friendly — Ideally: eventual integration into Coq codebase

Design goals

▷ User Friendly — Online learning, minimal configuration, works everywhere
▷ Installation Friendly — No external dependencies
▷ Integration Friendly — Coq plugin, with minor modifications to Coq
▷ Maintenance Friendly — Ideally: eventual integration into Coq codebase

Downside: We potentially sacrifice the use of some awesome Machine Learning algorithms

Minimum Viable Product$^{\text{TM}}$

Minimum Viable Product$^{\text{TM}}$

All components function!

Minimum Viable Product$^{\text{TM}}$

All components function! Kind off...

Minimum Viable Product^{TM}

All components function! Kind off...

Demo time

System Components

- Proof Recording
- Tactic Prediction
- Proof Search
- Proof Reconstruction

Proof Recording:

Proof Recording:

Ltac, Mtac, SSreflect, ML-tactics, Ltac 2.0?

Proof Recording:

Ltac, Mtac, SSreflect, ML-tactics, Ltac 2.0?

$\Downarrow$

Backtracking proof monad

Proof Recording:

Ltac, Mtac, SSreflect, ML-tactics, Ltac 2.0?

$\Downarrow$

Backtracking proof monad

Seems impossible: How do we introspect the monad?

Proof Recording:

Ltac, Mtac, SSreflect, ML-tactics, Ltac 2.0?

$\Downarrow$

Backtracking proof monad

Seems impossible: How do we introspect the monad?

For now, only Ltac recording

```
Definition left_pad_spec c s n :
  {s’ | (forall i, i < n - length s -> get i s’ = Some c)
     /\ (forall i, get i s = get (i + (n - length s)) s’)}.
Proof.
exists (left_pad c s n). unfold left_pad.
split.
- intros; rewrite <- append_correct1;
  [ rewrite cycle_get
  | rewrite cycle_length]; auto.
- intros; etransitivity;
  [ apply append_correct2
  | rewrite cycle_length; auto].
Qed.
```

```
Definition left_pad_spec c s n :
  {s’ | (forall i, i < n - length s -> get i s’ = Some c)
     /\ (forall i, get i s = get (i + (n - length s)) s’)}.
Proof.
record (exists (left_pad c s n)). record (unfold left_pad).
record (split).
- record (intros); record (rewrite <- append_correct1);
  [ record (rewrite cycle_get)
  | record (rewrite cycle_length)]; record (auto).
- record (intros); record (etransitivity);
  [ record (apply append_correct2)
  | record (rewrite cycle_length); record (auto)].
Qed.
```

```
c : ascii
s : string
n, i : nat
H : i < n - length s
______________________________________(1/1)
get i (cycle (n - length s) c ++ s) = Some c
```

```
rewrite <- append_correct1.
```

```
c : ascii
s : string
n, i : nat
H : i < n - length s
______________________________________(1/2)
get i (cycle (n - length s) c) = Some c
______________________________________(2/2)
i < length (cycle (n - length s) c)
```

```
[ascii]
[string]
[nat]
H : i < n - length s
______________________________________(1/1)
get i (cycle (n - length s) c ++ s) = Some c
```

```
rewrite <- append_correct1.
```

```
[ascii]
[string]
[nat]
H : i < n - length s
______________________________________(1/2)
get i (cycle (n - length s) c) = Some c
______________________________________(2/2)
i < length (cycle (n - length s) c)
```

```
[ascii]
[string]
[nat]
[le-i, le-minus, minus-n, minus-length, length-s]
______________________________________(1/1)
[eq-append, eq-Some, append-get, append-s, get-i, get-cycle, get-c, ...]
rewrite <- append_correct1.

[ascii]
[string]
[nat]
[le-i, le-minus, minus-n, minus-length, length-s]
______________________________________(1/2)
[eq-get, eq-Some, get-i, get-cycle, cycle-minus, minus-n, minus-length, ...]
______________________________________(2/2)
[le-i, le-length, length-cycle, length-c, cycle-minus, cycle-n, ...]
```

```
[ascii, string, nat,
le-i, le-minus, minus-n, minus-length, length-s, ...,
eq-append, eq-Some, append-get, append-s, get-i, get-cycle, get-c, ...]
rewrite <- append_correct1.

[ascii, string, nat,
le-i, le-minus, minus-n, minus-length, length-s,
eq-get, eq-Some, get-i, get-cycle, cycle-minus, minus-n, minus-length, ...,
le-i, le-minus, minus-n, minus-length, length-s]
```

intros	[eq-minus, minus-length, minus-max, max-x, max-y, length-t, ...]
intros	[nat, bool, plus-n, plus-length, length-k, eq-plus, eq-n]
⋮	⋮
rewrite sub_diag	[nat, eq-minus, eq-zero, minus-n, minus-n]
rewrite sub_diag	[nat, list, eq-minus, eq-zero, minus-length, minus-length, ...]
⋮	⋮
rewrite append_correct1	[ascii, string, nat, le-i, le-minus, minus-n, minus-length, ...]
⋮	⋮

? | [string, nat, eq-plus, eq-n, plus-length, plus-get, ...]

⇕

intros	[eq-minus, minus-length, minus-max, max-x, max-y, length-t, ...]
intros	[nat, bool, plus-n, plus-length, length-k, eq-plus, eq-n]
⋮	⋮
rewrite sub_diag	[nat, eq-minus, eq-zero, minus-n, minus-n]
rewrite sub_diag	[nat, list, eq-minus, eq-zero, minus-length, minus-length, ...]
⋮	⋮
rewrite append_correct1	[ascii, string, nat, le-i, le-minus, minus-n, minus-length, ...]
⋮	⋮

Machine Learning goal: Order database in order of relevance

k-nearest neighbor

Machine Learning goal: Order database in order of relevance

k-nearest neighbor

$$\text{Metric}: d(v_1, v_2) = \sum_{f \in v_1 \cap v_2} \log \frac{|D|}{|\{v \in D \mid f \in v\}|}$$

Machine Learning goal: Order database in order of relevance

k-nearest neighbor

$$\text{Metric} : d(v_1, v_2) = \sum_{f \in v_1 \cap v_2} \log \frac{|D|}{|\{v \in D \mid f \in v\}|}$$

$$\text{Jaccard} : d(v_1, v_2) = \frac{|v_1 \cap v_2|}{|v_1 \cup v_2|}$$

Machine Learning goal: Order database in order of relevance

k-nearest neighbor

$$\text{Metric} : d(v_1, v_2) = \sum_{f \in v_1 \cap v_2} \log \frac{|D|}{|\{v \in D \mid f \in v\}|}$$

$$\text{Jaccard} : d(v_1, v_2) = \frac{|v_1 \cap v_2|}{|v_1 \cup v_2|}$$

$$\text{Cosine} : d(v_1, v_2) = \frac{|v_1 \cap v_2}{\sqrt{|v_1||v_2|}}$$

Machine Learning goal: Order database in order of relevance

k-nearest neighbor

$$\text{Metric} : d(v_1, v_2) = \sum_{f \in v_1 \cap v_2} \log \frac{|D|}{|\{v \in D \mid f \in v\}}$$

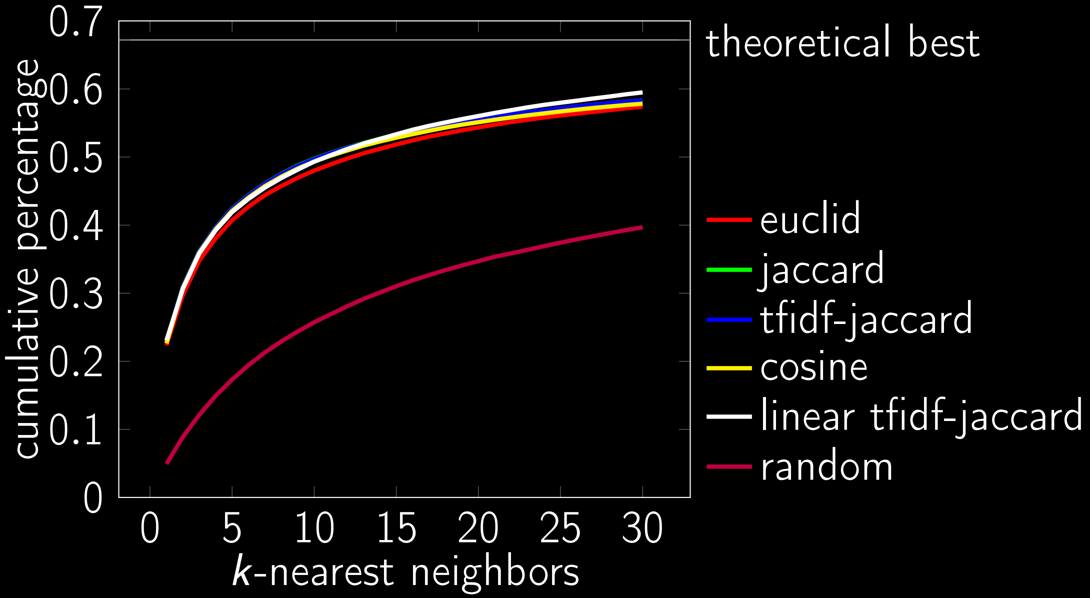
$$\text{Jaccard} : d(v_1, v_2) = \frac{|v_1 \cap v_2|}{|v_1 \cup v_2|}$$

$$\text{Cosine} : d(v_1, v_2) = \frac{|v_1 \cap v_2}{\sqrt{|v_1||v_2|}}$$

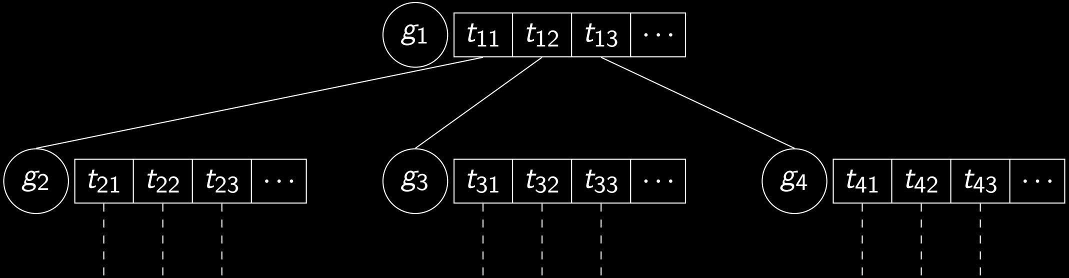
$$\text{Euclid} : d(v_1, v_2) = |v_1 \cup v_2 - v_1 \cap v_2|$$

Evaluation on Coq Standard Library: 144115 recorded pairs

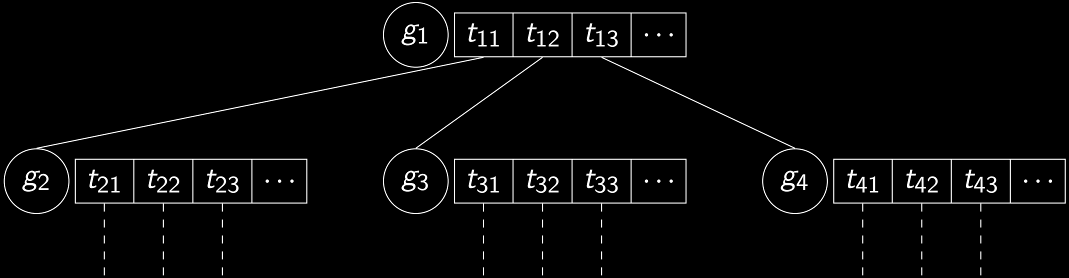
Evaluation on Coq Standard Library: 144115 recorded pairs



Proof Search



Proof Search



Skewed Breadth First Search

Let $t_1, \ldots, t_n$ be an ordered list of predicted tactics for goal g.
Subtree t_i is always explored one step deeper than subtree t_{i+1}.

Evaluation on Coq Standard Library

10778 lemmas, 2099 proved

19.5% Proved

Possible improvements

- Monte Carlo Tree Search
- Better Tactic Decomposition
- Better Feature Engineering
- Tactic Argument Prediction
- ...

?

- What did I do wrong
- What can I improve
- Innovative ideas?