Guiding SMT Solvers with Monte Carlo Tree Search and Neural Networks

Stéphane Graham-Lengrand   Michael Färber

28 March, 2018

Introduction

Inference calculus vs search control

Schulz, 2017

Improving heuristics has been the main source of progress in proof search!"

De Moura - Passmore, 2013

We present a challenge to the SMT community: to develop methods through which users can exert strategic control over core heuristic aspects of SMT solvers. We present evidence that the adaptation of ideas of strategy prevalent both within the Argonne and LCF theorem proving paradigms can go a long way towards realizing this goal.

Psyche

- Development by SGL
- Architecture that strongly separates inference calculus from search control
- Adaptation of the LCF approach where kernel's internal state is modified by search control primitives until an answer is found (SAT/UNSAT, Provable/Unprovable):
  trusted kernel ensures correctness
- Application to SMT-solving
  via the Conflict-Driven Satisfiability paradigm (CDSAT):
  lifts conflict-driven clause learning (CDCL) from SAT to SMT
- Handles multiple theories by making a modular list of "agents" cooperate

Modular agents for CDSAT

- contribute background knowledge, such as for Boolean logic or linear arithmetic
- offer for each state a set of possible decision assignments (e.g. assign truth value to literal $l \mapsto \texttt{true}$ or rational value to rational variable $x \mapsto {}^{3}\!/\!_{4}$)
- compute consequences of assignments (e.g. $x + y < 4$ and $x \geq 3$ implies $y < 1$)
- detect conflicts (e.g. $x < 4$ and $x > 6$)

CDSAT Main Loop

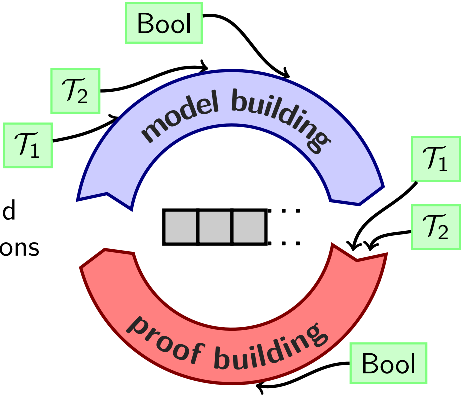
1. Assign values to terms/literals (model building)
2. If no conflict occurs: model exists (SAT)
3. If conflict occurs: analyse conflict and learn lemma (proof building)
   - 3.1 If learnt lemmas contradict: proof exists (UNSAT)
   - 3.2 Else: revert assignments and backtrack to 1

Data Structures

- Trail of assignments
- Learnt lemmas

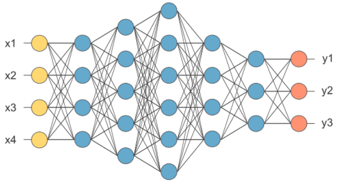
Statistics

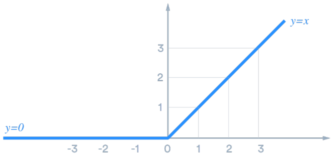
- 15–4000 decisions/second
- 100–3000 possible decisions each time



All theories at the same level is good - Example

Verification of deep neural nets with ReLU activation functions





("Is there an input satisfying P such that the output satisfies Q?")
The internal machinery is Linear Arithmetic + "if then else".
In theory, can be decided by off-the-shelf SMT-solver.
In practice, traditional SMT-solver would first split on "if then else" before theory reasoning (because SAT-solver is in the driving seat), and would therefore timeout.
CDSAT offers more flexibility: giving priority to Linear Arithmetic decisions rather than Boolean ones speeds up search.
With greater flexibility comes greater need for strategies.

Motivation for AI

Traditional SMT: Lots of hand-crafted heuristics

Do we want to keep on designing new hand-crafted heuristics for new kinds of decisions?

Psyche

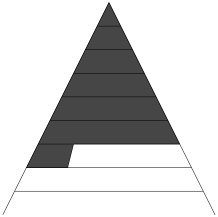
- $\oplus$ side: separates inference calculus from search control
  - answers are correct-by-construction, therefore
  - search control possibilities can be explored ad libitum (1)
- $\ominus$ side: performance not competitive with state of the art
  - suspected runtime overhead due to separating architecture
  - suspected runtime overhead due to purely functional kernel
  - no clever heuristics implemented so far; (1) not exploited

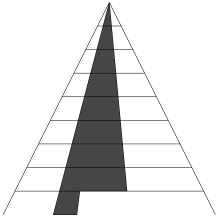
Hope

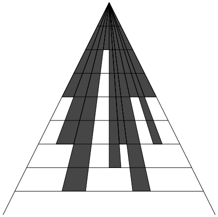
AI for heuristical guidance can handle new kinds of decisions, exploit (1), and somewhat improve performance.

Monte Carlo Tree Search

Illustration


(a) Iterative deepening without restricted backtracking.


(b) Iterative deepening with restricted backtracking.


(c) Monte Carlo.

Related Work

- AlphaGo (Silver et al., 2016)
- AlphaZero (Silver et al., 2017)
- Chemical Synthesis Planning (Segler et al., 2017)
- Monte Carlo Proof Search (Färber et al., 2017)

Monte Carlo Tree Search – Iteration

Monte Carlo Tree T: tree of visited states

1. Pick state s_1 among leaves of T using UCT, based on:
   - previous reward (exploitation)
   - number of traversals (exploration)
   - exploration constant: the higher, the more exploration
2. Play random moves (simulation): $s_1 \rightarrow s_2 \rightarrow \cdots \rightarrow s_n$
3. Calculate reward of s_n.
4. Add s_2 as child of s_1 in T (expansion).
5. Update rewards of all ancestors of s_2 in T.

- How to bias random moves?
- How to calculate reward of a state?

CDSAT as MCTS Problem

States

- local (MCTS) state: trail of assignments
- global state: learnt lemmas

Moves

- possible decision assignments
- given by modular agents
- simulation is performed until hitting conflict state

Backtracking

native CDSAT backtracking is replaced by MCTS

SAT/UNSAT and Exploitation/Exploration

SAT
search for a single state (needle in a haystack)

UNSAT
explore to learn complementary lemmas (cutting search space)

Exploitation/Exploration

- Exploitation: reward SAT-promising states
- Exploration: bias random moves towards complementary lemmas for UNSAT
- Exploration constant balances search between SAT and UNSAT

Move Probability

Bias move probability with theory-agnostic heuristics

Activity score (VSIDS)

prefer assigning terms or literals that participated in recent conflicts

Reward

How to estimate proximity of state to SAT?

Supervised learning

1. Traditional:
   - SAT states: label with maximal reward
   - conflict states: label with Levenshtein distance between conflict and actual SAT state
2. TD (temporal difference) learning:
   label (frequently visited) states with their MCTS reward

State characterisation

- trail of assignments
- generated lemma (if conflict state)
- previous lemmas present at time of conflict

Learning State Reward

Vector Space Embedding of Formulas

Motivation

- Goal: machine-learn `Reward` function on states
- many ML methods work on vectors
- vector space embedding of formulas allows us to embed states

Deep Graph Embedding: FormulaNet

- used for premise selection in Wang et al., 2017
- represent formulas as graphs to abstract from variable names
- learn vector representation of graphs with neural networks

Graph Embedding

$$\forall x \exists y (P(x) \wedge Q(x, y)) \quad \longrightarrow$$

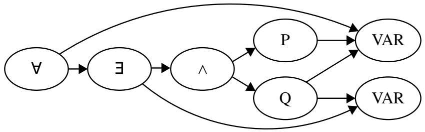


Figure 1: Making a graph *E* from a formula.

Vector Space Embedding (1)

Initialisation

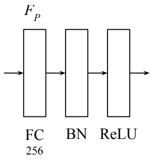
Every distinct node v in E is assigned distinct one-hot vector $\vec{x}_v^0$.

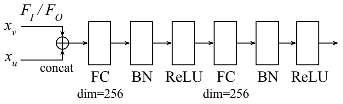
Update

Given a node v with degree d_v in E, its vector is updated as follows:

$$\vec{x}_v^{t+1} = F_P\left(\vec{x}_v^t + \frac{1}{d_v}\left[\sum_{(u,v)\in E} F_I(\vec{x}_u^t, \vec{x}_v^t) + \sum_{(v,u)\in E} F_O(\vec{x}_v^t, \vec{x}_u^t)\right]\right)$$

F_P, F_I, and F_O are realised by neural networks:





Vector Space Embedding (2)

Procedure

- perform *n* vector update steps
- max-pool all node embeddings to get graph embedding

Table 1: Validation accuracy of FormulaNet-basic on conditional premise selection for HolStep.

Number of steps	0	1	2	3	4
Accuracy	81.5	89.3	89.8	89.9	90.0

How to evaluate embedding quality during training?

- machine-learn `Reward` function using embedding
- evaluate with sum of cross-entropy losses over all update steps

Conclusion

Summary

Psyche

- inferences and search space well-identified
- prover states are persistent data structures (à la LCF) $\Rightarrow$ simplify recording of states and state switches during MCTS
- terms, formulas, trails etc. are constructed at most once during the run (hash-consing) $\Rightarrow$ use for efficient feature extraction?

MCTS

- bias search towards SAT/UNSAT via exploration/exploitation
- move probability via activity score (VSIDS)
- learn rewards either traditionally or via TD learning
- graph embeddings à la Wang

Project State

Already there

- Psyche
- Generic MCTS (from monteCoP)

both in OCaml

TODO

- integrate Psyche and MCTS modules
- implement vector space embedding of states with TensorFlow
- machine-learn `Reward` function using embedding
- organise training on example set