Proof Searching in PVS Theorem Prover using Simulated Annealing

M. Saqib Nawaz[1], Meng Sun[2] and **P. Fournier-Viger**[1]

[1]Harbin Institute of Technology (Shenzhen), China

[2]Peking University, Beijing, China

ICSI 2021

Qingdao, China

17-21 July, 2021

Introduction

- **Formal verification:**
  - verification of software and hardware systems using mathematical based techniques
  - e.g. safety-critical systems

- **Two main approaches**:
  - Model Checking
  - Theorem Proving

Introduction

- **Theorem Proving**:
  - a system is **modeled and specified** using mathematical logic.
  - **Important system properties verified** using *theorem provers.*
- Two main types:
  - **Automated theorem provers (ATPs)** based on FOL: computer programs that can perform logical reasoning.
  - **Interactive theorem provers (ITPs)** based on HOL: human interaction with the computer in the process of proof searching and development.
    Also known as proof-assistants.
- Well known proof assistants: PVS, HOL4, Coq.

Problem Statement

- The user guides the proof process in ITPs by providing the proof goal and applying proof commands and tactics to prove it.
- A user often does repetitive work to prove nontrivial tasks.
- For example, in PVS, a theory often contains too much information. It is inefficient to apply **brute force** or **pure random search** approach for proof searching.
- This makes proof guidance and proof automation along with proof searching some extremely desirable features for ITPs.
- A **proof searching approach** is proposed for the linkage between heuristic-based algorithms, such as **Simulated Annealing** with ITPs, such as **PVS**.

Prototype Verification System (PVS)

PVS consists of :

- a specification language (based on Higher Order Logic),
- a prover (based on Sequent Calculus),
- a parser,
- a type-checker, and
- specification libraries

Factorial function in PVS:

```
factorial(n:nat): RECURSIVE posnat =
IF n = 0 THEN 1 ELSE factorial(n-1)*n ENDIF
MEASURE n
the: THEOREM FORALL(k:nat): factorial(k) >= k
```

Related Work

Title	Approach	Data	Limitations
Hazel Duncan. (2007) **The use of data-mining for the automatic formation of Tactics**	Genetic Programming (Crossover operator)	Isabelle proofs	Proof represented as tree, which are linearized and lose information
Li-An Yang et al. (2016) **Automatically proving mathematical theorems with evolutionary algorithms and proof assistants**	GAs	Coq proofs	Successfully find the proofs of easy theorems (that contain a small number of proof steps) For large and complex theorems, interaction of users with Coq proof assistant is still required
Szu-Yi Huang and Ying-ping Chen. (2017) **Proving theorems by using evolutionary search with human involvement**	GAs		

Proof Searching Methodology

Simulated Annleaning (SA) is used for proof searching in PVS formalized theories.

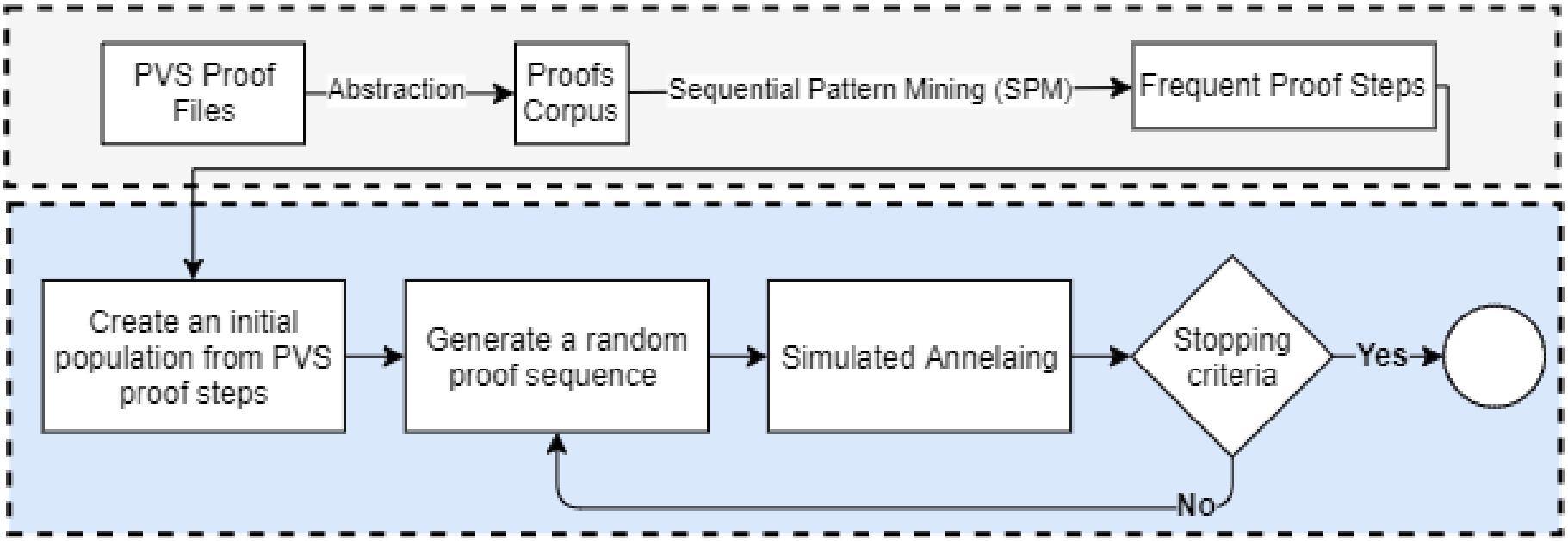

Table 1. A sample proof dataset

ID	Proof sequence
1	$\langle\{inst\ ,\ grind\}\rangle$
2	$\langle\{skosimp,\ expand,\ flatten,\ assert\}\rangle$
3	$\langle\{skosimp,\ expand,\ propax\}\rangle$
4	$\langle\{skeep,\ expand,\ typepred,\ expand,\ flatten,\ expand,\ inst\ ,\ assert\}\rangle$
5	$\langle\{induct\}$,$\{expand,\ propax\}$,$\{skosimp,\ expand,\ assert\}\rangle$

Proof Searching Methodology

Example: Specification for sum in PVS

```
sum(n): RECURSIVE nat = IF n=0
THEN 0 ELSE n+sum(n1)
ENDIF MEASURE n
```

Theorem

```
the: THEOREM sum(n) = (n*(n+1))/2
```

Proof steps used to prove the theorem

```
induct "n" expand "sum" propax
skosimp expand "sum" + assert
```

Extracted Proof Steps

1. induct
2. expand
3. propoax
4. skosimp
5. expand
6. assert

Fitness Score = 6

SA proof searching

Input: *FPPS*: Frequent PVS proof steps, Temp_min, α *PD*: proof sequences database, Temp,
Output: Generated proof sequences

1: $Pop \leftarrow FPPS$
2: **for each** $P \in PD$ **do**
3: $OF \leftarrow Fitness(P, P)$
4: $PS \leftarrow$ randomseq(Pop, length(P))
5: $BF \leftarrow Fitness(PS, P)$
6: **if** $BF \geq OF$ **then**
7: return PS
8: **end if**
9: **while** $(Temp > Temp\_min)$ **do**
10: $NS \leftarrow get\_neighbor(PS)$
11: $NF \leftarrow Fitness(NS, P)$
12: **if** $NF == OF$ **then**
13: return NS
14: **end if**
15: **if** $NF > BF$ **then**
16: $PS \leftarrow NS$
17: $BF \leftarrow NF$
18: **end if**
19: $ar \leftarrow exp(\frac{T}{1+T})$
20: **if** $ar >$ randomuniform$(0, 10)$ **then**
21: $PS \leftarrow NS$
22: $BF \leftarrow NF$
23: **end if**
24: $Temp \leftarrow Temp \times \alpha$
25: **end while**
26: return PS
27: **end for**

Proof Searching Methodology

Target Proof steps

```
1. induct
2. expand
3. propoax
4. skosimp
5. expand
6. assert
```

Fitness Score= 6
(Optimal)

Randomly Generated Proof Steps

```
1. expand
2. inst
3. grind
4. skosimp
5. typepred
6. Skolem!
```

Fitness Score= 1

Proof steps after Get Neighbor

```
1. skosimp
2. expand
3. grind
4. skosimp
5. typepred
6. Skolem!
```

Fitness Score= 2

Get Neighbor

Input: P_1: A proof sequence
Output: A neighbor of P_1

1: **procedure** GN(P_1)
2: $ind \leftarrow$ randomint(1, length(P_1))
3: $alter \leftarrow$ randomsample(Pop, 1) ▷ (*1-length* proof sequence form Pop)
4: $P_1[ind] \leftarrow alter$ ▷ ($P_1[ind] \neq$ alter)
5: return P_1
6: **end procedure**

Proof Searching Methodology

- Another main concept of SA is acceptance rate (AR).
- SA can select a new solution that is not better than previous solutions with the probability:

$$\mathrm{AR} = \exp\left(\frac{temp}{1+temp}\right)$$

- The temperature (Temp) is decreased in every iteration based on the following formula:

$$\mathrm{Temp} = \mathrm{temp} \times \alpha$$

- This probability helps SA to avoid local optimums, and explore other solutions.

Results

- We investigate the performance of the proposed SA for finding the proofs of theorems, lemmas and proof obligations in two PVS theories.
- 41 proof sequences and 23 distinct PVS proof steps (PPS)
- The SA was run ten times on the theorems.

Results of SA with AR		Results of SA without AR	
Avg. Gen. Count	Time (s)	Avg. Gen. Count	Time (s)
41, 569	0.71	42,134	0.72

- Result: acceptance rate (AR) does not play a major role in generation count or time.

Results

- Comparison of **SA** with a Genetic algorithm (**GA**) with different crossover and mutation operators.

Crossover operators:

- single point crossover (*SPC*),
- multi point crossover (*MPC*),
- uniform Crossover (*UCO*))

Two mutation operators:

- standard mutation (*SM*),
- modified pair-wise interchange mutation (*MPIM*)

Method	Avg. Gen. Count	Time (S)	Mem. (Mb)
SA	**41,569**	**0.71**	3,614
GA(SPC/SM)	902,772	21.06	**3,555**
GA(MPC/SM)	851,609	20.67	4085
GA(UCO/SM)	916,746	21.78	3,660
GA(SPC/MPIM)	182,679	9.85	3,617
GA(MPC/MPIM)	177,697	9.21	3,579
GA(UCO/MPIM)	173,055	9.05	3,618

- GA with different crossover and MPIM operator is approximately 5 times faster than the GA with different crossover operator and SM.
- SA is four times faster than GA with MPIM and different crossover operators

Statistical Test Results

Two hypotheses are tested:

- **H0**: *The means for the generations taken by the proof searching approaches for each proof sequence in the dataset are same.*
- **H1**: *The mean number of generations for at least one algorithm is different from the others.*

Friedman test indicate that the null hypothesis (H0) should be rejected with result of 187.41.

Wilcoxon p-value matrix for generations

	SPC/SM	MPC/SM	UC/SM	SPC/MPIM	MPC/MPIM	UC/MPIM	SA
SPC/SM	----	4.88E-1	7.90E-1	2.61E-8	2.42E-8	2.42E-8	2.42E-8
MPC/SM	4.88E-1	-----	4.88E-1	2.42E-8	2.42E-8	2.81E-8	2.42E-8
UC/SM	7.90E-1	4.88E-1	-----	2.61E-8	2.42E-8	2.42E-8	2.42E-8
SPC/MPIM	2.61E-8	2.42E-8	2.61E-8	-----	8.91E-1	4.64E-1	1.02E-5
MPC/MPIM	2.42E-8	2.42E-8	2.42E-8	8.91E-1	-----	6.30E-1	6.05E-7
UC/MPIM	2.42E-8	2.81E-8	2.42E-8	4.64E-1	6.30E-1	-----	2.46E-7
SA	2.42E-8	2.42E-8	2.42E-8	1.02E-5	6.05E-7	2.46E-7	-----

- Wilcoxon Test indicates that SA is significantly faster than different versions of GA.

Conclusion

- User interaction is required in ITPs to guide and find proof for particular goal.
  - Cumbersome and time consuming
- Proof searching approach is provided for the possible linkage between SA with HOL4.
- The performance of SA is compared with GA.
  - SA is much faster than GA

Future directions

- Making the proof searching process more general
  - Evolve frequent proof steps to a **compound** proof strategies for guiding the proofs of new **conjectures**.
- Use Curry-Howard isomorphism that provides a direct relation between **programming** and **proofs**.
  - With such correspondence, finding proofs can be viewed as writing programs.
  - A SA or GA, can be used to write programs (proofs) and PVS or other ITP for simplification and verification by computationally evaluating the programs.

THANKS!