Metaheuristic Algorithms for Proof Searching in HOL4

M. Saqib Nawaz[1], M. Zohaib Nawaz[2], Osman Hasan[3] and Philippe Fournier-Viger[1]
[1]School of Computer Science and Software Engineering, Shenzhen University, China
[2]Department of Computer Science and IT, University of Sargodha, Sargodha, Pakistan
[3]School of Electrical Engineering and Computer Science, National University of Sciences and Technology (NUST), Islamabad, Pakistan
msaqibnawaz@szu.edu.cn, zohaib.nawaz@uos.edu.pk, osman.hasan@seecs.edu.pk, philfv@szu.edu.cn

***Abstract*—User guided proof development in interactive theorem proving is a manual and time consuming activity. For automating proof searching and optimization in a higher-order logic proof assistant, we provide two metaheuristic algorithms that are based on Fitness Dependent Optimizer (FDO) and Bat Algorithm (BA). In both metaheuristic algorithms, random proof sequences are first created from a population of frequently occurring proof steps that are discovered using pattern mining techniques. Created proof sequences are then evolved till their fitness matches the fitness of the original (or target) proof sequences. Experiments are performed to investigate the performance of the proposed algorithms on different HOL4 theories. Moreover, the proposed FDO and BA-based proof searching approaches are compared with Simulated Annealing (SA) and Genetic Algorithm (GA)-based methods. Results show that BA performs best, followed by FDO and SA for proof finding and optimization in HOL4.**

***Index Terms*—Proof Searching, HOL4, Fitness Dependent Optimizer, Bat Algorithm, Simulated Annealing, Genetic Algorithm**

I. Introduction

Interactive theorem provers (ITPs) are used not only for the formalization of mathematical theorems and substantial parts of theoretical computer science, but also to model and verify complex software and hardware systems [2]. In ITPs, the systems that need to be analyzed are first modeled using an appropriate mathematical logic. Important (and sometimes critical) system properties are then proved using *theorem provers* [6]. ITPs are generally based on higher-order logic (HOL). The rich logical formalisms offered by HOL enable ITPs to define and reason about complex systems. However, due to the undecidability in HOL, the reasoning process cannot be made fully automated and human guidance is always required in the process of proof searching and development [11]. Due to this, ITPs are also called *proof assistants*. Some widely used proof assistants are HOL4 [19], Isabelle/HOL [16], Coq [3] and PVS [17].

The user driven proof development process makes the proof guidance and automation as well as automatic proof searching some extremely desirable features for ITPs. Evolutionary and heuristic algorithms, also indicated in [7], [9], [20], can be used to efficiently search for the proofs of theorems/lemmas because of their ability and suitableness to handle black-box search and optimization problems. Thus for the HOL4 proof assistant, we proposed an evolutionary approach [15], where a Genetic Algorithm (GA) was used for proof searching and optimization. Moreover, a simulated annealing (SA)-based proof searching approach was developed [13], which outperformed the GA-based method.

Both proof searching approaches [13], [15] were found to be quite efficient in evolving random proofs. However, alternative proof searching approaches could be developed as a plethora of evolutionary/heuristic techniques are present in the literature. Thus, we further investigate the applicability of evolutionary/heuristic techniques in the HOL4 proof assistant. This paper extends prior works [13], [15] by proposing two more metaheuristic-based approaches, where Fitness Dependent Optimizer (FDO) [1] and Bat Algorithm (BA) [21] are used for proof searching and optimization in HOL4. The performance of FDO and BA are compared with that of GA [15] and SA [13] for various parameter values. Through experiments on proof sequences of formalized theorems/lemmas in different HOL4 theories, it is found that BA performs better than the other three algorithms, followed by FDO and SA. Whereas, different versions of GA perform poorly for proof finding and optimization.

II. Related Work

Some work has been done in the past where evolutionary algorithms were used in ITPs. For example, a GA was used to automatically find the formal proofs of theorems/lemmas in the Coq proof assistant [7], [20]. But a major limitation of this approach is that even though it can find small proofs for theorems that contain a few proof steps, a user is still required to interact with Coq to guide the proof process for large theorems/lemmas that contain more proof steps. The work [9] briefly discussed how evolutionary computation can be used to improve the heuristics of automatic proof search in Isabelle/HOL. The objective is to find heuristics that can select the most promising PSL [10] (proof strategy language

DOI reference number: 10.18293/SEKE2022-103

for Isabelle/HOL) strategy from various available hand written strategies when applied to a given proof goal.

Another work [4] used genetic programming [8] and a pairwise combination (that focused only on crossover-based approach) to evolve frequent proofs patterns in the Isabelle proof assistant into compound tactics. However, a linearized tree structure was used to represent Isabelle's proofs. The linearization sometimes leads to the loss of important connections (information) among different branches in the proof trees. Due to this, the evolution process may not find interesting patterns and tactics in those trees.

The proposed proof searching approaches, presented in this paper, overcome the aforementioned limitations as they can handle proof goals of various lengths. Moreover, the dataset of proof sequences has all the important information that is needed for the determination of frequent proof steps, through which an initial population is generated. Lastly, the proposed approaches do not require any sort of human guidance in the evolution process for random proof sequences.

III. Proposed Proof Searching Approaches

HOL4 follows the interactive proof development process using the *lambda calculus proof representation*. Formal proofs in HOL4 can be constructed with an interactive *g*oal stack that are then put together using the ML function *prove*. A user of HOL4 interacts with the proof assistant to guide the proof process by providing necessary tactics, definitions, and already verified theorems. HOL4 also offers automatic proof procedures that help the user in directing the proof.

To use evolutionary/heuristic algorithms for proof searching and optimization, the data available in HOL4 proof files is first converted to a proper computational format. Moreover, the redundant information (related to HOL4) that plays no part in proof searching and evolution is removed from the proof files. Now, the complete proof for a theorem/lemma is a sequence of *HPS* (HOL4 proof step).

Let PS = $\{HPS_1, HPS_2, \ldots, HPS_m\}$ denote the set of *HPS*. PSS, a *proof step set*, is a set of *HPS*, i.e., $PSS \subseteq PS$. For example, consider that PS = {*DISCH_TAC, REPEAT_GEN_TAC, RW, FULL_SIMP_TAC, PROVE_TAC, REWRITE_TAC*}. The set {*FULL_SIMP_TAC,RW,DISCH_TAC,REWRITE_TAC*} is a proof step set that contains four *HPS*. A proof sequence is a list of PSS's, i.e., $S = \langle PSS_1, PSS_2, ..., PSS_n\rangle$, such that $PSS_i \subseteq PSS$ $(1 \leq i \leq n)$. For example, ⟨{*FULL_SIMP_TAC, PROVE_TAC*}, {*DISCH_TAC, REPEAT_GEN_TAC, REWRITE_TAC*}, {*RW*}⟩ is a proof sequence containing three PSS and six *HPS*.

A *proof dataset* PD is a list of proof sequences, i.e., $PD = \langle S_1, S_2, ..., S_p\rangle$. Each sequence in the PD has an identifier (ID) denoted as p. For example, Table I shows a *PD* that has five proof sequences.

A. Proposed FDO

The Fitness dependant optimizer (FDO) [1] is motivated by the swarming behavior of bees during reproduction when they explore and look for new hives. FDO consists of two processes: (1) The scout bee searching process, and (2) The scout bee movement process. In the first process, the scout bees search for a suitable solution. In the second process, a random walk and a fitness weight is used to move a scout bee towards a new position that indicates a potentially better solution.

Table I
A sample proof dataset

ID	Proof Sequence
1	⟨{*GEN_TAC, Q_TAC, SUFF_TAC*}⟩
2	⟨{*Q_TAC, SRW_TAC, HO_MATCH_MP_TAC*}⟩
3	⟨{*RW, AP_TERM_TAC, MAP_EVERYTHING_TAC, CONJ_TAC, PROVE_TAC*}⟩
4	⟨{*CASES_TAC, DISCH_TAC, SUBGOAL_THEN, CASES_ON, BETA_TAC, AP_TERM_TAC, GEN_TAC*}⟩
5	⟨{*SRW_TAC, Q.SUBGOAL_THEN, SUBST1_TAC, RW_TAC, Q.EXISTS_TAC, FULL_SIMP_TAC*}⟩

Algorithm 1 shows the pseudocode of the proposed FDO for proof finding and optimization in the HOL4 theories. FDO first creates an initial population (*Pop*) from frequent *HPS* (*FHPS*) that are discovered with sequential pattern mining (SPM) techniques [5]. From the initial population, a random scout bee (SB) is generated. The fitness of the solution, a target proof sequence (P), and SB is calculated with the fitness procedure listed in Algorithm 2. The fitness values guide the FDO towards the best solution(s) (proof sequences). Fitness evaluates the closeness (or similarity) of a given solution (SB) with the best solution (the target solution). The fitness value in this work denotes the total number of those positions in *SB* and P where *HPS* are same.

In FDO, the general equation to calculate the movement of a scout bee is:

$$x_{i,t+1} = x_{i,t} + pace \tag{1}$$

where $x_{i,t}$ represents the current scout bee at iteration (t) and the movement rate is denoted by $pace$ that sets the scout bee direction. The fitness value (fw) manages the $pace$:

$$fw = \left| \frac{x^*_{i,t,f}}{x_{i,t,f}} \right| \times wf \tag{2}$$

where $x*_{i,t,f}$ and $x_{i,t,f}$ denote the best global fitness of the solution, and the current fitness of the scout bee and wf is a weight factor that can be either 0 or 1. In our case, the best global fitness is the fitness of the target proof sequence (P) and the current fitness is the fitness of the current scout bee (SB). Moreover, wf is set to 1 as FDO was unable to evolve random scout bees to target proof sequences when wf = 0.

FDO considers some scenarios for fw to provide a random mechanism for the $pace$. For example, if fw = 1 or 0, and $x_{i,t,f}$ = 0, FDO uses Equation (3) to find the $pace$ randomly. On the other hand, if $0 < fw < 1$, then FDO generates a random number r, in the range [-1, 1], to make sure that the scout bee searches in every direction. For different values of r, the $pace$ is calculated using equation (4):

$$pace = (x_{i,t} \times r) \qquad if((fw = 1 \wedge 0) \wedge x_{i,t,f} = 0) \tag{3}$$

Algorithm 1 FDO proof finding

Input: *FHPS*: Frequent HOL4 proof steps, *PD*: proof sequences database
Output: Generated proof sequences

```
Pop ← FHPS
for each P ∈ PD do
    g_b ← Fitness(P, P)
    SB ← randomseq(Pop, length(P))
    p_b ← Fitness(SB, P)
    FS ← ()
    if p_b = g_b then
        return SB
    end if
    while (p_b < g_b) do
        for i in range(length(SB)) do
            if SB[i] = P[i] then
                FS.append[i]
            end if
        end for
        Calculate movement with pace using Equation (5)
        NS ← update_position(SB, pace, FS)
        NF ← Fitness(NS, P)
        if NF = g_b then
            return NS
        end if
        if NF > p_b then
            SB ← NS
            p_b ← NF
        end if
    end while
    return SB
end for
```

$$pace = \begin{cases} (x_{i,t} - x*_{i,t})fw(-1) & \text{if } 0 < fw < 1 \vee r < 0 \\ (x_{i,t} - x*_{i,t})fw & \text{if } 0 < fw < 1 \vee r \geq 0 \end{cases} \tag{4}$$

Algorithm 2 Fitness

Input: *Pseq*: A proof sequence, *P*: The current target proof sequence
Output: Integer that represents the fitness of a proof sequence (*Pseq*)

```
procedure FITNESS(Pseq, P)
    i, f ← 0
    while (i ≤ length(Pseq) - 1) do
        if (Pseq[i] = P[i]) then
            f ← f + 1
        end if
        i ← i + 1
    end while
    return f
end procedure
```

According to the nature of our problem, the movement in Equation (1) is adapted to be an integer number. Thus, Equation (1) is modified as:

$$x_{i,t+1} = x_{i,t} + \lfloor pace \rfloor \tag{5}$$

where $\lfloor pace \rfloor$ returns the integer that is less than or equal to $pace$.

Equation (5) for updating the movement basically indicates how many positions are required to be changed in a random proof sequence (SB) so that it reaches the next position. In the position update process, randomly selected position values in a scout bee are changed from their original values.

Algorithm 3 Update_Position

Input: PS: A proof sequence, UP: updated position, and FS: $fixedSlots$ array
Output: Updated Sequence

```
procedure UPDATE_POSITION(PS, UP, FS)
    for i in range(0, UP) do
        rp ← randomint(1, length(PS))
        if (rp ∉ FS) then
            alter ← randomsample(Pop, 1)      ▷ (1 HPS form Pop)
            PS[rp] ← alter                    ▷ (PS[rp] ≠ alter)
        end if
    end for
    return PS
end procedure
```

We use an array called *fixedSlots*(FS) to further enhance the searching process in FDO. This array keeps track of each position in SB that has matched its value with the P. During the update of SB positions, it is checked whether each position in SB that is to be replaced with a random *HPS*, is already present in *FS* or not. If the position is present in FS, then another random number is generated for a different position. If the position is not present, then that particular position is updated with a random HPS from *Pop*. Algorithm 3 is the procedure for updating the position.

B. Proposed BA

Bat Algorithm (BA) [21] is inspired by the echolocation behavior of microbats, with varying pulse rates of emission and loudness. The three main steps of BA are: (1) Estimating the optimal distance of bats towards the solution(s) using the phenomena of echolocation, (2) Bats moving in the search space with distinct velocity and fixed frequency. The wavelength and loudness can vary according to bats distance between solution(s) and the bat current position, and (3) Linearly decreasing the loudness and increasing the emission factor of bats when they are near to the solution(s). Algorithm 4 shows the pseudocode of the proposed BA for proof searching and optimization in HOL4 theories.

BA also creates an initial population (*Pop*) first from *FHPS*. A random proof sequence, that represents a bat (B), is then generated from the population. In general, the BA uses the following equations to calculate the frequency, velocity, and position of a bat, respectively:

$$f_i = f_{min} + (f_{max} - f_{min})\beta \tag{6}$$

$$v_i^{t+1} = v_i^t + (x_i^t - X_*)f_i \tag{7}$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{8}$$

where f_i is the frequency of the i-th bat, f_{max} and f_{min} represent the maximum and minimum frequencies of the sound waves released by bats, and β is a random number in the range [0, 1]. Moreover, v_i^t and v_i^{t+1} represent the velocities of the i-th bat at iterations t and t + 1, respectively, x_i^t and x_i^{t+1}

Algorithm 4 BA proof finding

Input: *FHPS*: Frequent HOL4 proof steps, *PD*: proof sequences database
Output: Generated proof sequences

Pop $\leftarrow$ *FHPS*
for each $P \in PD$ **do**
  $g_{best} \leftarrow Fitness(P, P)$
  $B \leftarrow$ randomseq(*Pop*, length(*P*))
  $p_{best} \leftarrow Fitness(B, P)$
  $FS \leftarrow ()$
  if $p_{best} = g_{best}$ **then**
    return B
  end if
  while ($p_{best} < g_{best}$) **do**
    for i in range(length(B)) **do**
      if $B[i] = P[i]$ **then**
        $FS.append[i]$
      end if
    end for
    Calculate updated velocity (v_i^{t+1}) using Equation (11)
    $NS1 \leftarrow update\_position(B, v_i^{t+1}, FS)$
    Calculate y_i^{t+1} using Equation (12)
    $ran1 \leftarrow rand(0, 1)$
    if $ran1 < y_i^{t+1}$ **then**
      $NS \leftarrow$ Neighbor($NS1$)
    else
      NS $\leftarrow$ NS1
    end if
    $NF \leftarrow Fitness(NS, P)$
    if $NF = g_{best}$ **then**
      return NS
    end if
    if $NF > p_{best}$ **then**
      $B \leftarrow NS$
      $p_{best} \leftarrow NF$
    end if
  end while
  return B
end for

represent the locations of the i-th bat at iterations t and t + 1, respectively, and X_* represents the current optimal best location. In this work, X_* is equal to the total number of *HPS* in the target proof sequence.

While approaching towards the prey (target proof sequence in this work), a bat increases its pulse emission rate and decreases its loudness. These phenomena are simulated using the following equations:

$$A_i^{t+1} = \alpha A_i^t \tag{9}$$

$$r_i^{t+1} = r_i^0(1 + exp(\gamma t)) \tag{10}$$

where A_i^t and A_i^{t+1} represent the loudness at iterations t and t + 1, respectively, r_i^0 represents the initial pulse emission rate, r_i^{t+1} represents the pulse emission rate at iteration t + 1 and $0 < \alpha < 1$ and $\gamma > 0$ are constants.

For proof searching and optimization, a random bat (B) updates its velocity, location, loudness, and pulse emission rates repeatedly, until the target proof sequence (P) is reached.

Similar to movement update for FDO, the velocity update Equation (7) for BA is rewritten as:

$$v_i^{t+1} = v_i^t + \lfloor (gBest - x_i^t) f_i \rfloor \tag{11}$$

where $gBest$ is equal to X_* in Equation (11) and $\lfloor (gBest - x_i^t) f_i \rfloor$ returns the integer that is less than or equal to $(gBest - x_i^t)$.

Equation (11) for updating the velocity in BA indicates how many positions are required to be changed in a random proof sequence (B) so that it reaches the next position. Algorithm 3 is also used in BA to update the velocity position.

Equations (9) and (10) are used to calculate the loudness and pulse emission of a bat (B). Here, we combine these two equations together as follows:

$$y_i^{t+1} = A_i^{t+1} + r_i^{t+1} \tag{12}$$

Using the BA idea, if a random number $ran1$ is less than y_i^{t+1}, then one position value is changed in bat B. Algorithm 5 lists the procedure for changing one position in B. The randomly selected location value (*HPS*) is changed from its original value using the $Neighbor$ procedure in Algorithm 5.

Algorithm 5 Neighbor

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

procedure NEIGHBOR(P_1)
  $ind \leftarrow$ randomint(1, length(P_1))
  $alter \leftarrow$ randomsample(*Pop*, 1)   ▷ (*1-HPS* form Pop)
  $P_1[ind] \leftarrow alter$ ▷ ($P_1[ind] \neq$ alter)
  return P_1
end procedure

IV. Results and Discussion

The proposed FDO and BA are implemented in Python[1]. To evaluate the proposed approaches, experiments were carried on a computer equipped with a fifth generation Core $i5$ processor and 4 GB of RAM. For FDO, the weight factor wf is set to 1. For BA, f_{min} and f_{max} are set to 0 and 10, respectively. Whereas, r_i^0, the initial loudness (A_i^0), α and γ are set to 0.2, 1, 0.8 and 0.9, respectively.

Table II
A SAMPLE OF THEOREMS / LEMMAS IN SOME HOL4 THEORIES

HOL Theory	No.	HOL4 Theorems / Lemmas
Transcendental	L1	`⊢ ∀x. 0<=x∧x <= inv(2) ==> exp(x) <= 1+2*x`
	T1	`⊢ ∀ x. (\n. (^exp_ser) n*(x pow n)) sums exp(x)`
	T2	`⊢ ∀ x. 0 < x∧ x < 2 ==> 0 < sin (x)`
Arithmetic	T3	`⊢ ∀n a b. 0 < n ==>((SUC a MOD n = SUC b MOD n) = ( a MOD n = b MOD n ))`
RichList	T4	`⊢ ∀m n. ((l:'a list). ((m + n)=(LENGTH l))==> ( APPEND ( FIRSTN n l ) ( LASTN m l ) = l)`
Number	T5	`⊢ ∀n m. ( m <= n ==> (iSUB T n m = n - m)) ∧ (m < n ==> (iSUB F n m = n - SUC m))`
	T6	`⊢ ∀ n a. 0 < onecount n a ∧ 0 < n ==> ( n = 2 EXP (onecount n a - a ) - 1 )`
Sort	T7	`⊢(PERM L[x]<==>(L= [x])∧(PERM [x] L <==>(L = [x])`
	T8	`⊢ PERM = PERM_SINGLE_SWAP`
Rational	T9	`⊢ ∀ x y. abs_rat ( frac_add (rep_rat ( abs_rat x)) y ) = abs_rat ( frac_add x y )`
	T10	`⊢ ∀ r1 r3. rat_les r1 r3 ==> ?r2. rat_res r1 r2 ∧ rat_les r2 r3`

[1]Code available at: https://github.com/saqibdola/BAFDO-HOL4

We examined the performance of proposed FDO and BA in finding the correct proofs of theorems/lemmas in 14 different HOL4 theories available in its library. These theories are: *Transcendental, Arithmetic, RichList, Number, Sort, Rational, Bool, FiniteMap, InductionType, BinaryWords, Encode, Coder, Decode* and *Combinator*. From each theory, five to twenty theorems/lemmas were randomly selected. The *PD* contains 300 proof sequences in total and 93 distinct *HPS*. Some important theorems/lemmas from aforementioned theories are listed in Table II. Table III shows the performance of the FDO and BA on theorems/lemmas that are listed in Table II.

Recently, we used a GA [15] with various crossover and mutation operators and a SA [13] for proof searching and optimization in HOL4. Just like FDO and BA, an initial population for GA and SA was first created using the SPM-based learning approach [14]. Crossover and mutation operators were used in GA and annealing process in SA to evolve the random proof sequences towards the original (target) proofs. In GA, three crossover operators (single point crossover (SPC), multi point crossover (MPC) and uniform Crossover (UCO)) and two mutation operators (standard mutation (SM) and modified pairwise interchange mutation (MPIM)) were used. The main reason to use different versions of crossover and mutation operators was to compare their effect on the overall performance of GAs in proof searching.

We run the algorithms for GA and SA on *PD* to compare its performance with FDO and BA. The comparison of FDO, BA with SA and GA for the theorem ($T2$) is shown in Table III. For $T2$, BA performed better (8,017 generations) than others, followed by FDO (16,767 generations) and SA (30,346). For GA, we found that using different crossover operators has no major effect on its overall performance. However, MPIM was faster to find the target proof sequences than SM.

Table III
RESULTS FOR FDO, BA AND COMPARISON WITH GA, SA

T/L	Algorithm	Fitness	Generations	Time (s)
L1	FDO	54	10,027	0.362
T1		58	10,809	0.385
T2		**81**	**16,767**	**0.793**
T3		66	11,887	0.475
T4		19	4,073	0.062
T5		23	4,858	0.071
T6		20	3,880	0.069
T7		17	3,186	0.0417
T8		42	6,959	0.166
T9		23	4,594	0.062
T10		23	4,436	0.060
L1	BA	54	5,885	0.251
T1		58	6,124	0.273
T2		**81**	**8,017**	**0.506**
T3		66	6,414	0.272
T4		19	1,974	0.052
T5		23	2,335	0.075
T6		20	2,921	0.080
T7		17	1,758	0.019
T8		42	4,487	0.098
T9		23	3,012	0.078
T10		23	2,894	0.082
T2	SA	81	30,346	0.858
T2	GA(SPC/SM)	81	2,231,664	58.56
T2	GA(MPC/SM)	81	2,713,867	69.84
T2	GA(UC/SM)	81	2,905,410	75.63
T2	GA(SPC/MPIM)	81	500,500	14.89
T2	GA(MPC/MPIM	81	524,272	16.14
T2	GA(UC/MPIM)	81	589,292	17.15

The average number of generations for the four algorithms to reach the target proof sequences in the whole dataset are shown in Table IV. BA performed better than other algorithms, followed by FDO, SA and GA. The possible reason for this is that the $fixedSlots$ array in FDO and BA ensures that no changes are made in those positions where the HPS in both random solutions (bee in FDO and bat in BA) and the target solution (original proof sequence) match. This prevents the mismatching of HPS at already matched positions in both solutions. The reason for BA performing better than FDO is that besides the update velocity function, the random bat in BA also goes through the $Neighbor$ procedure that allows more diversity. GA with different crossover and MPIM operators is approximately fourteen times faster (generation wise) than GA with different crossover operators and SM. This is due to the fact that the SM changes the *HPS* at a single location in the sequence, whereas MPIM changes two locations. Thus, MPIM explores a more diverse solution as compared to SM. On the other hand, SA is six times faster than GA with MPIM and different crossover operators. Whereas, FDO is approximately one and seven times faster than SA, and BA is approximately twice faster than FDO. Moreover, the memory used by proof searching approaches while searching for proofs of formalized theorems/lemma in *PD* is also listed in Table IV. All the algorithms require approximately the same memory with negligible difference.

Table IV
AVERAGE TOTAL GENERATION COUNT FOR FDO, BA, SA AND GA

	Avg. Generation Count	Total Time	Memory
FDO	779,819	14.15 s	3521 Mb
BA	375,044	9.09 s	3454 Mb
SA	1,319,745	19.54 s	3459 Mb
GA(SPC/SM)	123,513,780	1844.50 s	3545 Mb
GA(MPC/SM)	120,580,649	1697.47 s	3463 Mb
GA(UC/SM)	119,633,993	1569.69 s	3507 Mb
GA(SPC/MPIM)	8,833,888	194.25 s	3550 Mb
GA(MPC/MPIM)	9,141,943	208.34 s	3702 Mb
GA(UC/MPIM)	8,704,233	190.491 s	3682 Mb

The longest proof in the *PD* is for theorem $T2$ (positive value of sine) and it consists of 81 *HPS*. Here we call this theorem *PVoS*. We checked how much time the four algorithms take generation wise and also how many correct *HPS* in *PVoS* are found in different generations by the algorithms. The lines in Figure 1 represent the time for algorithms and the bars represent the fitness achieved by the algorithms. BA reached the maximum fitness of 81 within approximately 8,000 generations. As generation increases, the performance of algorithms tend to decrease for fitness. This means that with more generations, algorithms were slow in finding the correct *HPS* for a proof sequence as compared to earlier generations. An interesting behavior for GA is that it tends to decrease the fitness values (found *HPS*) in some generations. For example, GA(MPC/MPIM) and GA(MPC/SM) found less *HPS* at 11,000 and 13,000, respectively, compared to correctly found *HPS* at earlier locations (10,000 and 12,000 respectively). The other algorithms do not exhibit such behavior.

Lastly, the four algorithms are compared in terms of convergence speed to examine how fast the algorithms were able to

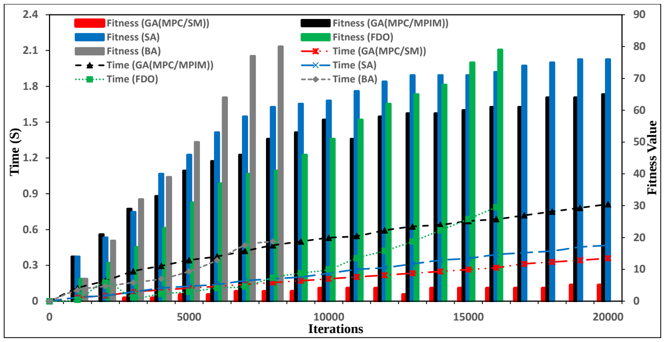

Figure 1. Time and fitness in different generations

converge towards the optimal solution. For the first 20,000 generations, the convergence speed of the four algorithms for *PVoS* is shown in Figure 2. BA converges very fast and found the correct *HPS* in approximately 8,000 generations. The performance of GA(MPC/MPIM) and SA was same compared to BA at the start. However, after 5,000 generations, GA(MPC/MPIM) and SA took more generations in finding the remaining correct HPS. The performance of SA after 13,000 generations tends to get low. Whereas, FDO performance was linear and fast from the beginning. On the other hand, GA(MPC/SM) convergence speed is slow from the start. At 20,000 generations, GA(MPC/MPIM) finds approximately 64 correct *HPS*, GA(MPC/SM) finds approximately 5 correct *HPS*, whereas SA finds 76 correct *HPS*.

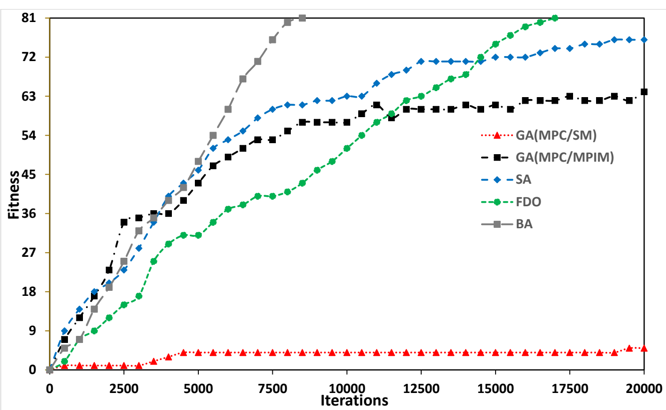

Figure 2. Convergence performance

In summary, it was observed through experiments that the proposed FDO-based and BA-based proof searching approaches can quickly optimize and automatically find the correct proofs for formalized theorems/lemmas in HOL4 theories. The proof searching approaches in this work and in [12], [13], [15] are not limited to HOL4 and can be used in proof assistants such as Isabelle/HOL [16], Coq [3], and PVS [17].

V. Conclusion

Despite huge developments in the last two decades, ITPs still depend on user interaction to manually guide proof assistants in finding the proof for a conjecture (unproved theorem or lemma). This interaction makes the proof development process quite complicated and a time consuming activity for the users. This paper introduced two proof searching approaches based on FDO and BA for optimizing and finding the correct proofs in various HOL4 theories. Additionally, a performance comparison of the two approaches with SA and GA showed that both FDO and BA performed better than them.

In future, we are interested in exploiting the Curry-Howard correspondence in sequent calculus [18] that offers a relationship between programming and mathematical proofs. This will allow us to use evolutionary/heuristic techniques to write programs (proofs) and use HOL4 proof assistant to simplify and verify by computationally evaluating the programs.

References

[1] J. M. Abdullah and T. Ahmed. Fitness dependent optimizer: Inspired by the bee swarming reproductive process. *IEEE Access*, 7:43473–43486, 2019.
[2] J. Avigad, J. C. Blanchette, G. Klein, L. C. Paulson, A. Popescu, and G. Snelting. Introduction to milestones in interactive theorem proving. *Journal of Automated Reasoning*, 61(1-4):1–8, 2018.
[3] Y. Bertot and P. Casteran. *Interactive theorem proving and program development: Coq'Art: The calculus of inductive construction*. Springer, 2003.
[4] H. Duncan. *The use of data-mining for the automatic formation of tactics*. PhD thesis, University of Edinburgh, UK, 2007.
[5] P. Fournier-Viger, J. C. W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas. A survey of sequential pattern mining. *Data Science and Pattern Recognition*, 1(1):54–77, 2017.
[6] O. Hasan and S. Tahar. Formal verification methods. In *Encyclopedia of Information Science & Technology, 3rd edition*, pages 7162–7170. IGI Global, 2015.
[7] S. Y. Huang and Y. P. Chen. Proving theorems by using evolutionary search with human involvement. In *Proceedings of CEC 2017*, pages 1495–1502. IEEE, 2017.
[8] J. R. Koza. *Genetic programming - On the programming of computers by means of natural selection*. MIT Press, 1993.
[9] Y. Nagashima. Towards evolutionary theorem proving for Isabelle/HOL. In *Proceedings of GECCO (Poster) 2019*, pages 419–420. ACM, 2019.
[10] Y. Nagashima and R. Kumar. A proof strategy language and proof script generation for Isabelle/HOL. In *Proceedings of CADE 2019*, volume 10395 of *LNCS*, pages 528–545. Springer, 2017.
[11] M. S. Nawaz, M. Malik, Y. Li, M. Sun, and M. I. U. Lali. A survey on theorem provers in formal methods. *CoRR*, abs/1912.03028, 2019.
[12] M. S. Nawaz, M. Z. Nawaz, O. Hasan, P. Fournier-Viger, and M. Sun. An evolutionary/heuristic-based proof searching framework for interactive theorem prover. *Applied Soft Computing*, 104:107200, 2021.
[13] M. S. Nawaz, M. Z. Nawaz, O. Hasan, P. Fournier-Viger, and M. Sun. Proof searching and prediction in HOL4 with evolutionary/heuristic and deep learning techniques. *Applied Intelligence*, 51(3):1580–1601, 2021.
[14] M. S. Nawaz, M. Sun, and P. Fournier-Viger. Proof guidance in PVS with sequential pattern mining. In *Proceedings of FSEN 2019*, volume 11761 of *LNCS*, pages 45–60. Springer, 2019.
[15] M. Z. Nawaz, O. Hasan, M. S. Nawaz, P. Fournier-Viger, and M. Sun. Proof searching in HOL4 with genetic algorithm. In *Proceedings of SAC 2020*, pages 513–520. ACM, 2020.
[16] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A proof assistant for higher-Order logic*. Springer, 2002.
[17] S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. PVS system guide, PVS prover guide, PVS language reference. Technical report, SRI International, November 2001.
[18] J. E. Santo. Curry-howard for sequent calculus at last! In *Proceedings of TLCA 2015*, volume 38 of *LIPIcs*, pages 165–179, 2015.
[19] K. Slind and M. Norrish. A brief overview of HOL4. In *Proceedings of TPHOL 2008*, volume 5170 of *LNCS*, pages 28–32. Springer, 2008.
[20] L. A. Yang, J. P. Liu, C. H. Chen, and Y. P. Chen. Automatically proving mathematical theorems with evolutionary algorithms and proof assistants. In *Proceddings of CEC 2016*, pages 4421–4428. IEEE, 2016.
[21] X. Yang. A new metaheuristic bat-inspired algorithm. In *Nature Inspired Cooperative Strategies for Optimization (NICSO)*, pages 65–74, 2010.