MAX-SAT Problem using Evolutionary Algorithms

H. M. Ali, David Mitchell[‡], and Daniel C. Lee
School of Engineering Science, School of Computing Science[‡]
Simon Fraser University
Burnaby BC, Canada
(hmali, dgm, dchlee)@sfu.ca

***Abstract*—MAX-SAT is a classic NP-hard optimization problem. Many real problems can be easily represented in, or reduced to MAX-SAT, and thus it has many applications. Finding optimum solutions of NP-hard optimization problems using limited computational resources seems infeasible in general. In particular, all known exact algorithms for MAX-SAT require worst-case exponential time, so evolutionary algorithms can be useful for finding good quality solutions in moderate time. We present the results of an experimental comparison of the performance of a number of recently proposed evolutionary algorithms for MAX-SAT. The algorithms include the Artificial Bee Colony (ABC) algorithm, Quantum Inspired Evolutionary Algorithm (QEA), Immune Quantum Evolutionary Algorithm (IQEA), Estimation of Distribution Algorithm (EDA), and randomized Monte Carlo (MC). Our experiments demonstrate that the ABC algorithm has better performance than the others. For problems with Boolean domain, such as MAX-SAT, the ABC algorithm requires specification of a suitable similarity measure. We experimentally evaluate the performance of the ABC algorithm with five different similarity measures to indicate the better choice for MAX-SAT problems.**

Keywords—MAX-SAT; evolutionary algorithm; Optimization; CNF;

I. INTRODUCTION

Boolean Satisfiability (SAT) is the problem of deciding if a Boolean formula in conjunctive normal form (CNF) has a satisfying truth assignment. As usual, formulas are constructed from a set of n Boolean variables, x_1 through x_n, with connectives for conjunction ($\wedge$), disjunction ($\vee$), and negation ($\bar{x}$). An SAT instance, or CNF formula, is a conjunction of clauses, where each clause *is* a disjunction of literals and each literal, l is an affirmation (x_i) or negation ($\overline{x_i}$) of a variable $x_i \in \{x_1, \ldots, x_n\}$. A solution for a CNF formula is an assignment mapping each of its variables to true (1) or false (0), in such a way that at least one literal in each clause is made true. An example of CNF is:

$$(x_1 \vee \overline{x_2}) \wedge (x_2 \vee \overline{x_3}) \wedge (x_2 \vee x_4) \wedge (x_4 \vee \overline{x_1})$$

A satisfying assignment for the above CNF formula is: $x_1 = 1$, $x_2 = 1$, $x_3 = 0$, and $x_4 = 1$. A CNF formula may have zero, one or more satisfying assignments.

The Maximum Satisfiability (MAX-SAT) problem is an optimization variant of SAT. In MAX-SAT, the problem is to find a truth assignment that maximizes the number of true clauses. We can view truth assignments as strings or vectors over a binary alphabet containing 0 and 1. MAX-SAT is then the problem of finding string that maximizes the following function:

$$f(x) = \sum_{i=1}^{c} S_i(x) \quad (1)$$

In the above equation, $S_i(x)$ is true (1) if i is satisfied by assignment, x, and false (0) otherwise. If a literal of the clause is true then the clause true.

A large variety of NP-hard problems can be modeled or represented as SAT or MAX-SAT, and solving other problems by reduction to one of these is widely studied. Problems frequently solved this way include model-checking of finite state systems, design debugging, AI planning, routing, scheduling, bio-informatics [1-3], etc.

The literature contains a variety of algorithms for MAX-SAT, which can be characterized as providing either exact solutions or approximate solutions. Among algorithms intended for practical use, the exact algorithms are mainly based on branch and bound (BnB) algorithm [3], and differ from each other in variable selection heuristics, inference rules, etc. The approximate techniques do not guarantee the optimal solution, but these approaches can provide reasonably good quality results with limited resources (i.e., time, memory, computation, etc.). Evolutionary computation is one of the popular approximation techniques for obtaining good quality solutions in with limited resources. In the past, Genetic Algorithm (GA) or variant of GA such as GASAT and pEvoSAT have been evaluated on different MAX-SAT problems [16]. In this study, we evaluate several recently proposed evolutionary algorithms on MAX-SAT instances: the Artificial Bee Colony Algorithm (ABC), Immune Quantum Evolutionary Algorithm (IQEA), Quantum Inspired Evolutionary Algorithm (QEA) and Estimation of Distribution Algorithm (EDA) [6-9].

In this study, our benchmark instances are formulas produced by the *sgen* (Standard Generator) formula generator. *Sgen* aims to generate small and difficult benchmark instances for SAT, and has been used elsewhere for MAX-SAT evaluations well. This software has been presented in the annual SAT competition 2011, and it fulfills input requirements for that competition.

This work is supported in part by HEC (Higher Education Commission) and Islamia University Bahawalpur (IUB) Pakistan, under faculty development program, and in part by the Natural Sciences and Engineering Research Council of Canada.

978-1-4799-4458-3/14/$31.00 ©2014 IEEE

The rest of the paper is organized as follows. Section II describes the evolutionary algorithms we study. Section III discusses the experimental design. Section IV presents and discusses the results. Section V provides the conclusion.

II. EVOLUTIONARY ALGORITHMS

Intelligent behaviors of nature are an inspiration for the Evolutionary algorithms (EAs). In an EA, candidate solutions, or points in the search space, for an optimization problem are considered to be individuals in a population. The definition of an individual to the problem is a key to efficiently utilize the EAs [6]. For MAX-SAT, we can simply take truth assignments candidate solutions (individuals), and these can be viewed as finite Boolean vectors. The fitness of an individual in the population is the value of a cost function at that point.

A. Artificial Bee Colony Algorithm

The artificial bee colony (ABC) algorithm imitates the behaviors of the real bees in finding food source positions for their nectar. Bees explore a space looking for positions which are good food sources, and later communicate information about the food sources to the other bees in their hive using a waggle dance. In the ABC algorithm, a food source position represents a possible solution to the optimization problem (i.e., an individual in the population), and its nectar quality corresponds to the fitness function value. The model of foraging that represents the collective intelligence of a honey bee swarm is based on three components: food sources, employed bees, and unemployed bees. The model also involves two unique behaviors, the abandonment of a food source and the recruitment of a food source.

In ABC, each employed bee has been assigned to a food source, and knows its nectar quality or fitness. The algorithm has two types of unemployed bees: scout bees and onlooker bees. Scout bees are unemployed bees that explore the space looking for new food sources. Scout bees become employed again as soon as they locate a new food source with better nectar. Onlooker bees are young bees about to start their career. These bees receive information from employed bees during their visit to hives. In the algorithm, an onlooker bee selects a food source known to one of the employed bees, with a probability proportional to the nectar quality, and then becomes an employed bee. Finally, if no improvement is observed in a solution after some maximum number of algorithm trials (which is a parameter), the algorithm abandons that food source, and its associated employed bee becomes a scout. A scout selects a new random food source and tests its nectar quality. As soon as a scout bee is associated to a food source, it becomes an employed bee [12].

The ABC algorithm was originally designed for problems defined on continuous and integer domain [6, 8]. The MAX-SAT problem is a Boolean domain problem, and requires choice of a similarity suited for the Boolean domain [4]. We use five different similarity measures to see the efficiency of the ABC algorithm for MAX-SAT problem. These techniques include Simple Matching Similarity Measure (SMSM), Jaccard's Similarity Measure (JSM), Dice or Czekanowski Sorenson Similarity Measure (DCSS), Sokal and Sneath Similarity Measure 1 (SSSM1), and Rogers and Tanimoto Similarity Measure (RTSM) [4, 15].

In Binary ABC, there is only one employed bee for every food source, and they are *N* employed bees; hence, the number of solutions in the population is equal to the number of employed bees, *N*. An individual, or a food source position, is a vector of m binary numbers denoted by $x \triangleq (x_1, x_2, \dots, x_m)$, and the i^{Th} individual of the population is denoted by $x^i, i \in \{1, \dots, N\}$.

Let x^a and x^b be two binary vectors. Following are four possible cases to compare corresponding bits of the two vectors:

$x^a = x^b = 1$

$x^a = x^b = 0$

$x^a = 1, x^b = 0$

$x^a = 0, x^b = 1$

Given two binary vectors, x^a and x^b, we define the following functions:

- Z_{11} is the number of bits where both x^a and x^b have a value of 1
- Z_{00} is the number of bits where both x^a and x^b have a value of 0
- Z_{10} is the number of bits where x^a is 1 and x^b is 0
- Z_{01} is the number of bits where x^a is 0 and x^b is 1

For an individual of dimension m, $Z_{11}+Z_{00}+Z_{10}+Z_{01}=m$. The definitions of our five different similarity measures are as follows [4] [15]:

$$\text{Similarity}\,(x^a, x^b) = \frac{Z_{11}+Z_{00}}{Z_{11}+Z_{10}+Z_{01}+Z_{00}} \quad (2)$$

$$\text{Similarity}\,(x^a, x^b) = \frac{Z_{11}}{Z_{11}+Z_{10}+Z_{01}} \quad (3)$$

$$\text{Similarity}\,(x^a, x^b) = \frac{2(Z_{11})}{2(Z_{11})+Z_{10}+Z_{01}} \quad (4)$$

$$\text{Similarity}\,(x^a, x^b) = \frac{2(Z_{11}+Z_{00})}{2(Z_{11}+Z_{00})+Z_{10}+Z_{01}} \quad (5)$$

$$\text{Similarity}\,(x^a, x^b) = \frac{Z_{11}+Z_{00}}{Z_{11}+Z_{00}+2(Z_{10}+Z_{01})} \quad (6)$$

Equations (2) – (6) define SMSM, JSM, DCSS, SSSM1, and RTSM respectively. The value of each of these measures is always in the interval [0 1]. We define corresponding dissimilarity measures, also in [0 1], by [4]:

$$\text{Dissimilarity}\,(x^a, x^b) = 1 - \text{Similarity}\,(x^a, x^b) \quad (7)$$

Details of the mapping of dissimilarity measure to the original ABC function are available in [4]. The differential expression for binary ABC is represented as:

$$\text{Dissimilarity}\,(x^k, x^b) \approx \phi.\,\text{Dissimilarity}\,(x^a, x^b) \quad (8)$$

In equation 8, $x^k, x^a,$ and x^b are binary vectors and ϕ is a positive random scaling factor. Let

$G = \phi$Dissimilarity (x^a, x^b) with the value of G determined using any one of the equations from 2 to 6. x^k is a new binary solution, which we want to construct in such a way that its degree of dissimilarity with x^b is around the value of G. Therefore, the values of the four functions Z_{11}, Z_{10}, Z_{01}, and Z_{00} must be determined to form x^k as a candidate solution.

The degree of Dissimilarity (x^k, x^b) is obtained by equation (8) in such a way that its value should be around the value of G. Let n_1 be the total number of 1s, and n_0 is total number of value 0s in x^b. The following integer program gives us the best possible values for Z_{11}, Z_{10}, Z_{01} and Z_{00}. The system of equations is as follows [4]:

$$min \left| 1 - \frac{Z_{11}}{Z_{11}+Z_{10}+Z_{01}} - G \right| \quad (9)$$

Such that:

$$Z_{11}+Z_{01} = n_1 \quad (10)$$

$$Z_{10} \leq n_0 \quad (11)$$

$$Z_{11}+Z_{10}+Z_{01} \geq 0 \text{ : and integer} \quad (12)$$

The objective of this integer program (equation 9) is to minimize the gap between the values of Dissimilarity (x^k, x^b) and G. $Z_{11}+Z_{01}$ counts the number of bits in x^b where value is 1. Therefore, constraint 10 enforces this value must be equal to n_1. Constraint 11 ensures that total number of bits where x^k is 1 is and x^b is 0 must be less than the total number of zeroes in x^b (i.e., n_0).

Initially, ABC randomly generates N food source positions and sends scout bees. In this step, each bee finds the food source position and saves it in her memory and becomes an employed bee. The process then enters into algorithms' generation loop which contains three phases.

In the employed bees phase, each employed bee finds a new food source position x^b in the neighborhood of the current food source x^a, and checks if the new food source x^k has better nectar, saving its location if it does. The new food source x^k is generated using the similarity measures (i.e. equation 2 to 6).

During the onlooker bee phase, first employed bees share information about the quality of food sources with onlooker bees. An onlooker bee probabilistically chooses an employed bee to receive her food source information based on the following probability [6]:

$$P_s = \frac{(0.9 * F_s)}{Max(F)} + 0.1, \text{ where } s = 1,2, \ldots, N \quad (13)$$

Where, N is the size of the population and F_s is the fitness value of the s^{th} solution in the population. After an onlooker bee received the food source information (location and nectar value), she flies to the food source. At this time, since the onlooker bee is associated with the food source, she turns into an employed bee and locates a new food source in her neighborhood using the chosen similarity measure (one of equations 2 to 6). After that, she compares the nectar quality of the new food source with her associated food source, and saves the best location and nectar. This process can be thought of as exploitation in evolutionary algorithms. The ABC algorithm also uses the following exploration technique. If there is no improvement in the nectar quality of a food source after a maximum number of algorithm trials, the food source is abandoned and the employed bee assigned to that food source will become a scout bee, which then selects a random food source location.

Parameter definitions for the MAX-SAT problem for ABC algorithm are as follows [6]:

1. F denotes the objective function in equation 1,
2. N denotes the population size (number of food sources in ABC),
3. x denotes an individual (food source position in ABC),
4. g denotes the number of algorithm generations,
5. The similarity measure (an employed bee locates the new food source using a similarity measures among equations 2 to 6).
6. The probability used in equation 13, determining how bees greedily select an employed bee.
7. γ is the limit of the trail for scout bee phase,
8. I is the maximum number of generations.

Pseudo code for the MAX-SAT version of the ABC algorithm is as follows:

Initialization

Generate the population (i.e., sending scouts) of size N,
Define stopping criterion,
Define the limit trial γ,
Trial $T = 0$,

While (*Stopping criterion is not satisfied*)

Employed Bee Phase:

for N (Food Sources)

Each employed bee locates a new food source in the neighborhood of the current food source using the measure, checks the fitness F and saves the new food source position if it is better.
If food source not improved, increment Trail: $T = T + 1$,

end for

Probability defined by the formula in equation 13,

Onlooker Bee Phase:

OBC (Onlooker bee counter) = 0,

While (OBC < N (Food Sources))

if *(rand* > P_s *)*

Onlooker bee selects am employed bee and becomes an employed bee,
OBC = OBC + 1,
Repeat Employed Bee Phase,
Check the Fitness F, If food source not improved, increment in Trail: $T = T + 1$,

end if

end *While*

Scout Bee Phase:

if $(T > \gamma)$

Replace the food source which reaches the trial limit,

end if

Saves the Best fitness.

End While

B. Quantum Inspired Evolutionary Algorithm

In classical evolutionary algorithms, each individual is typically represented by a binary string. In the quantum inspired evolutionary algorithm (QEA), each individual is represented by a string of Q-bits, where a Q-bit is the probabilistic representation inspired by the qubit concept in the quantum computing. In quantum computing, the smallest unit of information is known as quantum bit (or qubit). Unlike classical computers that represent information in binary digits '0' and '1', the qubit may be in the state '1', state '0', or in any superposition of these two states. The state $|\psi\rangle$ of a qubit can be written as:

$$|\psi> = \alpha|0> + \beta|1> \quad (14)^1$$

where, α and β are in general complex numbers, $|\alpha|^2$ and $|\beta|^2$ are the probabilities that the qubit will be found in the "0" state and the "1" state, respectively [6]. The values of α and β may vary with time as the state evolves during the computation. However, these values must satisfy:

$$|\alpha|^2 + |\beta|^2 = 1 \quad (15)$$

In quantum computing, quantum gates are used to modify the values of α and β. A number of quantum gates are available in the literature for quantum computation -- e.g. XOR gate, NOT gate, Hadamard gate, rotation gate, etc. A quantum gate can be represented by a unitary operator (matrix) *U*. Matrix *U* is unitary if and only if $UU^{\dagger} = U^{\dagger}U = I$, where *I* is an identity matrix, and $U^{\dagger}$ denotes the conjugate transpose of *U*. The unitary matrix ensures that the probability values satisfy (15). Readers are referred to reference [13] for further details on quantum computing and QEA. We use the following rotation gate for QEA.

$$R = \begin{bmatrix} Cos(\theta) & -Sin(\theta) \\ Sin(\theta) & Cos(\theta) \end{bmatrix} \quad (16)$$

Parameter definitions for the MAX-SAT problem for QEA algorithm are as follows [6]:

1. *F* denotes the objective function,
2. *N* denotes the population size,
3. x denote an individual, a binary string which intends to make the maximum clauses of the given formula true.
4. *g* denotes the number of algorithm generations,
5. Define θ to be the rotation gate that changes the Q-bit string value, and $|\alpha|^2$ and $|\beta|^2$ to be the probabilities of "0" and "1" states, respectively.

Pseudo code for QEA implementation for the MAX-SAT problem is given as follows.

Initialization

Initialize the $\alpha_i^{j,g=0}$ and $\beta_i^{j,g=0}$ with $\frac{1}{\sqrt{2}}$ for *i=1,2,…,m*, and *j = 1,2…N,* (Initializing with $\frac{1}{\sqrt{2}}$ ensures there is no bias at *g=0*)

Initialize the rotation angle θ_i^j 0 for *i=1, 2… m*, and *j = 1, 2,…,N.*

Execution

While (*convergence criterion is not satisfied*)

Generate a population of *N* binary strings with $\alpha_i^{j,g}, \beta_i^{j,g}$ with $\frac{1}{\sqrt{2}}$ for *i=1,2,…,m*, and *j = 1,2…N,*,

Evaluate the fitness values of *N* individuals,

Store the best solution among *N*,

Save best individual from *N,*

Get the historically best individual, W_b=[$w_{b,1}, w_{b,2,\ldots,} w_{b,m}$], and the best individual from the current generation is C_b=[$c_{b,1}, c_{b,2,\ldots,} c_{b,m}$],

for *i = 1, 2,…,m*

if $w_{b,i} = 1$ *and* $c_{b,i} = 1$

$\theta_i = \omega$

elseif $w_{b,i} = 0$ *and* $c_{b,i} = 0$

$\theta_i = -\omega$

else

$\theta_i = 0$

Set $\theta_i^j = \theta_i$ for *j = 1, 2…, N,*

end *for*

Update $\alpha_i^{j,g}$, *and* $\beta_i^{j,g}$ using θ_i^j,

g = g +1,

End *while*

C. Immune Quantum Inspired Evolutionary Algorithm

Immune quantum evolutionary algorithm (IQEA) is an extension of QEA by adding an immune operator into QEA. Immune algorithms are evolutionary algorithms based on physiological immune systems. Physiological immune systems have mechanisms that enable cells to exhibit and recognize foreign substances and develop an immune operation. Immune operation based immune operator is employed in IQEA to improve QEA. Further study regarding the immune algorithm and its functionality is presented in [6]. The immune operator is based on two major operations: immune recognition and clonal selection.

Immune recognition is used for raising fitness. For an individual *i*, an immune recognition means modifying the genes on some bits in accordance with prior knowledge, to get higher fitness with greater probability. Suppose a population $\Omega = i_1, i_2, \ldots, i_k, \ldots, i_N$, the immune recognition on Ω means the operation carried out on ψ = nα individuals. These individuals are selected from Ω population in

[1] Bra-ket notation is a standard notation for describing quantum states in the theory of quantum computing. It is composed of angle brackets and vertical bars. $\langle a|$ is known as bra and $|a\rangle$ is known as ket.

proportion of α, where α is the probability of selecting individuals from population for immune operation.

Clonal selection is used for preventing deterioration. Clonal selection states that antigens can selectively react to antibodies; if antibody matches the antigen sufficiently well, and antibody's B cell (i.e., number of bits from certain individual) is stimulated and can produce related clones. In the traditional artificial immune clone algorithms scale of the copy of clone (i.e. the number of bits copied from one individual to other individual) is fixed. However, we adopt the dynamic copy of the clone as follows [6]:

$$D_i^g = round\left(\frac{N*fit(\Omega_i^g)}{\sum_i^N fit(\Omega_i^g)}\right) \quad (17)$$

Where, D is the size of the clone, N is the size of the population at generation g. If fitness of the offspring is greater (in context of MAX-SAT problem) then the fitness of parent, the offspring will participate in the next competition or vice versa.

Parameter definitions for the MAX-SAT problem for IQEA algorithm are as follows:

1. F denotes the objective function,
2. N denotes the population size,
3. x denotes an individual, a binary string, which intends makes the maximum clauses of the given formula true,
4. g denotes the number of algorithm generations,
5. Apply immune operator on selected individuals (i.e., nα).
6. Define θ to be the rotation gate that changes the Q-bit string value, and $|\alpha|^2$ and $|\beta|^2$ to be the probabilities of "0" and "1" states, respectively.

Pseudo code for IQEA implementation for the MAX-SAT problem is given as follows.

Initialization

Initialize the $\alpha_i^{j,g=0}$ and $\beta_i^{j,g=0}$ with $\frac{1}{\sqrt{2}}$ for $i=1, 2\ldots m$, and $j = 1, 2\ldots N$,

Initializing with $\frac{1}{\sqrt{2}}$ to ensure there is no bias at $g=0$, represents the algorithm generation,

Initialize the rotation angle θ_i^j 0 for $i=1, 2\ldots m$, and $j = 1, 2, \ldots, N$,

Initialize immune rate, $\alpha = 0.5$,

Execution

While (*convergence criterion is not satisfied*)

Generate a population of N binary strings with $\alpha_i^{j,g}, \beta_i^{j,g}$ with $\frac{1}{\sqrt{2}}$ for $i=1,2,\ldots,m$, and $j = 1,2\ldots N,$,

Evaluate the fitness values of N individuals,

Evaluate the fitness values of N individuals,

Store the best solution among N,

After sorting, choose ψ which defines immune recognition,

Apply colonel selection on each ψ individual(s) and use equation 12 to calculate dynamic scale of clone,

Make N by merging ψ

Save best individual from N,

Get the historically best individual, $W_b=[w_{b,1}, w_{b,2,\ldots,}w_{b,m}]$, and the best individual from the current generation is $C_b=[c_{b,1}, c_{b,2,\ldots,}c_{b,m}]$,

for $i = 1, 2, \ldots, m$

if $w_{b,i} = 1$ *and* $c_{b,i} = 1$

$\theta_i = \omega$

elseif $w_{b,i} = 0$ *and* $c_{b,i} = 0$

$\theta_i = -\omega$

else

$\theta_i = 0$

Set $\theta_i^j = \theta_i$ for $j = 1, 2\ldots, N$,

end *for*

Update $\alpha_i^{j,g}$, *and* $\beta_i^{j,g}$ using θ_i^j,

$g = g + 1$,

End *while*

D. Estimation of Distribution Algorithm

Estimation-of-Distribution Algorithm (EDA) is a population based probabilistic evolutionary algorithm (EA) [9]. In classical evolutionary algorithms like Genetic Algorithm (GA) manipulation of strings using crossover and mutation operators is the basic criteria of evolution. Each individual is a candidate solution in population. The fitness/objective value of each individual in the population is evaluated in the first phase of each generation. In the second phase, the individuals with higher fitness values are selected as the parents of the individuals for the next population. Unlike the GA, EDA produces a new population by evaluating probability density distribution rather than manipulating strings. EDA updates its population at each generation on the basis of probability densities obtained from superior candidates evaluated and chosen at the previous generation. Then, a new population of individuals is randomly generated from the probability distribution estimated from the previous generation. In this work, we adopt EDA with the univariate marginal distribution algorithms (UMDAs). More details on UMDA is available in [14].

Parameter definitions for the MAX-SAT problem for EDA algorithm are as follows [9]:

1. F denotes the objective function,
2. Ω denotes the population (set of N individuals), where N denotes the population size,
3. g is the maximum number of generations,
4. x denotes an individual, a binary string which intends to make the maximum clauses of the given formula true,
5. η_g is the set of the best candidate solution selected from the g^{th} generation,
6. P_e is the selection probability (for MAX-SAT we used 0.5) of the best individuals selected from the g^{th} generation,
7. Estimate the probability density distribution Γ from

the best individuals of g - 1th generation.

Pseudo code for EDA implementation for the MAX-SAT problem is given as follows.

Initialization

Generate new population randomly Ω of size *N*,
Define stopping criterion,

Execution

While (*Stopping criterion is not satisfied*)
Rate the individual of the current population by evaluating fitness function,
Select the best individuals, $P_e * (N$ individuals) from g - 1th generation,
Estimate the probability density distribution Γ on the basis of η_{g-1} best individuals,
Generate new population Ω on the basis of new probability distribution,

End While

III. Experiment Design

Figure 1 shows the general scheme for our evaluation evolutionary algorithms for the MAX-SAT problem. The scheme can be used with any evolutionary algorithm for MAX-SAT in which candidates are binary strings. We use the standard generator (Sgen) to produce small but challenging instances [1]. Sgen takes the number of variables and SAT/UNSAT as arguments to generate instances. The instances generated are CNF formulas with a given number of clauses over a given number of variables, in the standard DIMACS CNF file format. A utility CNF_READ reads the CNF file and produces an expression. Another utility accepts this expression and binary string equal to the size of number of variables, and returns the number of satisfied clauses.

The algorithm parameters used in the experiments are as follows.

Parameters for IQEA are as follows: $\theta = \frac{\pi}{15}$ and α = 0.5

Parameters for QEA: $\theta = \frac{\pi}{15}$

Parameters for ABC: $\gamma = 0.2$ and $\phi = 0.7$

For all algorithms, the population size is 30, the stopping criterion is the maximum number of function evaluations, and the number of replication is 10.

TABLE II. SIMILARITY MEASURES COMPARISON

Vars.	Opt.	SMSM	JSM	DCSS	SSSM1	RTSM
20	48	47	47	47	**48**	47
25	60	59	58	59	**59**	59
30	72	70	70	70	70	70
35	84	83	83	82	83	82
40	96	94	95	94	94	94
45	108	106	105	105	105	104
50	120	117	118	117	117	116
60	144	139	140	140	140	140
80	192	185	185	184	185	185
100	240	229	229	230	**231**	230
120	288	275	275	275	275	275
140	336	321	319	320	319	319
160	384	365	363	365	**365**	364
180	432	409	408	411	**410**	410
200	480	454	454	454	**455**	453

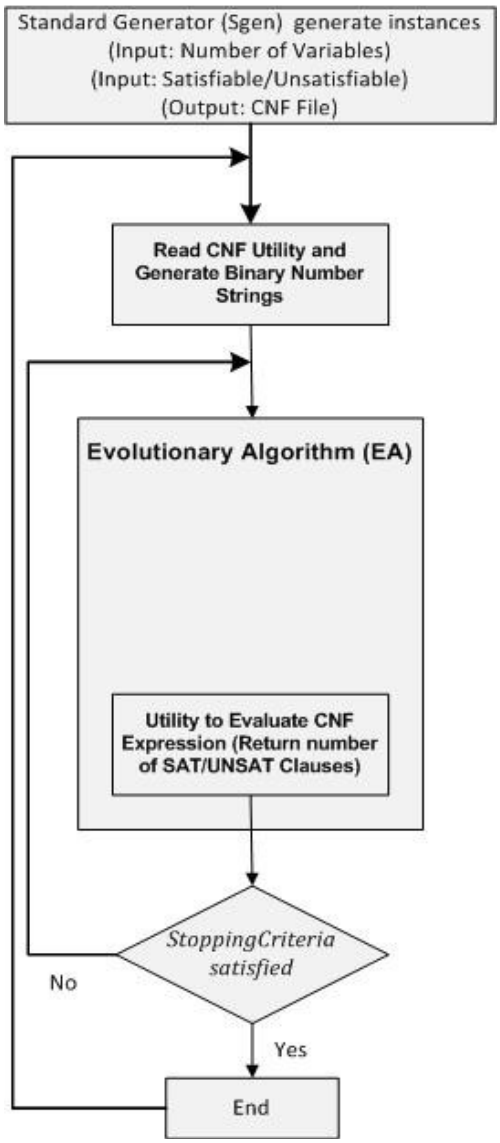

Fig.1. Experiment design for MAX-SAT using EAs

IV. Results and Discussion

Table III and figure 2 demonstrate the simulation results for the MAX-SAT problem using EAs. Results are tabulated for every algorithm and show the maximum fitness value (i.e., with header *Best*) achieved. *Opt.* represents total number of clauses in an instance and *Best* value shows the number of satisfied clauses. *Avg.* value illustrates the results of over 10 independent simulation runs for every algorithm. Standard deviation (i.e., *Std.*) shows how often an algorithm reaches closer or equal to the best value.

Algorithms achieving the highest value are considered the best for that particular number of variables. However, if two or more algorithms achieve the same best value the algorithm having the minimum standard deviation is considered as the best for that particular number of variables. The minimum standard deviation illustrates the confidence level of an

TABLE III. SIMULATION RESULTS

			ABC			EDA			IQEA			QEA			MC		
Vars.	Optm.	Max. Fun. Evl.	Avg.	Std.	Best	Avg.	Std.	Best	Avg.	Std.	Best	Avg.	Std.	Best	Avg.	Std.	Best
20	48	500	46.60	0.5164	47	45.10	0.8756	46	46.4	0.699	47	46.10	**0.316**	**47**	37.40	2.9515	42
25	60	500	58.00	0.0000	58	55.60	1.0750	57	57.1	0.875	58	57.40	**0.699**	**59**	48.30	3.8312	54
30	72	500	69.60	**0.5164**	70	65.20	1.3166	68	68.4	1.075	69	68.90	0.567	70	56.60	3.8355	63
35	84	8000	81.70	**0.6749**	83	76.70	1.3375	78	79.3	1.159	81	80.20	1.032	82	68.30	4.1110	74
40	96	8000	93.10	**0.8756**	95	87.30	1.6364	90	90.7	1.059	92	91.00	0.666	92	76.70	5.2292	83
45	108	8000	104.1	**0.5676**	105	96.60	2.0656	99	100.8	1.032	103	102.0	0.816	103	83.60	4.2999	89
50	120	8000	115.5	**0.9718**	118	106.9	2.4244	110	112.0	0.666	113	113.2	0.918	115	92.50	5.4620	101
60	144	20000	138.6	**0.6992**	140	127.7	2.790	133	134.5	0.707	136	135.6	1.264	138	112.0	6.4464	123
80	192	20000	183.6	**0.6992**	185	168.2	2.8983	172	178.7	0.823	180	179.8	1.229	182	149.0	8.9567	159
100	240	20000	227.9	**0.7379**	229	209.5	2.7588	213	221.4	1.173	223	223.7	1.494	227	186.6	8.3958	203
120	288	30000	273.4	**0.6992**	275	247.6	2.5473	252	264.1	2.233	267	267.7	1.494	271	229.3	9.8438	242
140	336	30000	317.8	**0.7888**	319	287.8	5.3914	299	306.4	2.716	312	311.1	2.331	316	254.9	18.070	293
160	384	30000	362.0	**0.6667**	363	330.5	5.3177	342	349.5	1.509	352	355.5	1.828	358	300.6	16.263	330
180	432	30000	406.9	**0.8756**	408	365.3	6.0928	373	389.4	3.272	394	398.4	3.025	404	337.7	16.042	361
200	480	30000	450.7	**1.3375**	454	407.0	6.4979	418	430.3	2.790	434	444.2	1.932	447	383.4	20.748	412

evolutionary algorithm to achieve best values or targets over repeated experiments. We increase the resources for every algorithm (e.g., increase in number of function evaluations) with the increase in dimension of the problem (i.e., increase in number of variables). However, results show the gap between the optimal values (i.e., number of clauses to satisfy in a MAX-SAT instance) and the best by an algorithm.

ABC, QEA and IQEA have the same maximum value for 20 variables. QEA is only better than every other algorithm in the set for 25 variables. QEA and ABC both have the same best value for the 30 variables, but ABC will consider a better algorithm, because of low standard deviation. After 30 variables, ABC is better than every other algorithm not only in terms of the best value, but also in standard deviation. This trend extends from 35 to 200 variables. Either for a small number of variables or for a large number of variables, EDA and Monte Carlo (MC – random algorithm) has the least efficiency, both in terms of best value, and low standard deviation as compared to the other algorithms considered for this study. Similarly, QEA is better than the IQEA. Although IQEA employed the immune operator to better utilize QEA's ability to explore and exploit the search space, it fails to surpass the QEA. Table II shows the comparative study for five similarity measures for MAX-SAT problems. By keeping in view these results, SSSM1 is considered a better choice than other counterparts. This study concludes that the ABC algorithm and SSSM1 similarity measure are better candidate choices for the class of MAX-SAT problems.

V. CONCLUSION

In this paper, we described several relatively new evolutionary algorithms (EAs) that can be used to find a high-quality solution for the class of MAX-SAT problems. Finding an exactly optimal solution for this type of problem becomes increasingly difficult as the size of the MAX-SAT problem instance increases. EAs can be useful for finding a high-quality suboptimal solution of the MAX-SAT problem with limited computing resources. The goal of this study is to experimentally compare evolutionary algorithms for this class of problems. These algorithms include the Artificial Bee Colony Algorithm (ABC), the Immune Quantum Evolutionary Algorithm (IQEA), the Quantum Inspired Evolutionary Algorithm (QEA), the Estimation of Distribution Algorithm (EDA), and the randomized Monte Carlo (MC) algorithm. Simulation results show that the ABC algorithm using the SSSM1 similarity measure is a better choices for the class of MAX-SAT problems. Our experiments conclude that the ABC algorithm using the SSSM1 similarity measure outperforms other competing counterparts, not only in achieving good quality solutions, but also with good confidence levels (i.e., low standard deviations).

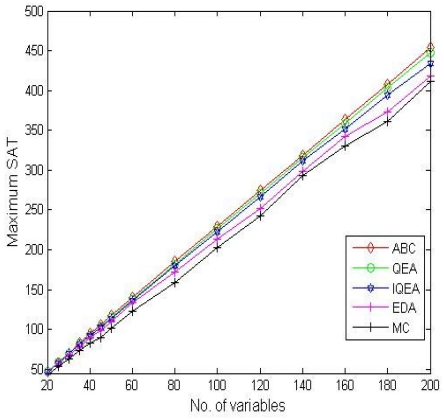

Fig. 2. EAs Results

References

[1] A. Kattan, A. and E. Galvan, "Evolving radial basis function networks via GP for estimating fitness values using surrogate models," *Proceedings of IEEE Congress on Evolutionary Computation (CEC)*, Brisbane, Australia, June 10 - 15, 2012

[2] N. Bouhmala, "Multilevel diversification and intensification in meta-heuristics" *Proceedings of 8th IEEE International Conference on Intelligent Systems: Theories and Applications (SITA)* Tonsberg, Norway, May 8 - 9, 2013

[3] A. Abrame, and D. Habet, "Inference Rules in Local Search for MAX-SAT," *Proceedings of IEEE 24th International Conference on Tools with Artificial Intelligence (ICTAI).* Marseille, France, November 7 - 9, 2012

[4] M. H. Kashan, Nahavandi, and A. H. Kashan, "DisABC: A new artificial bee colony algorithm for binary optimization." *ELSEVIER Journal, Applied Soft Computing,* vol. 12(1) Jan. 2012

[5] S. Malik, and Z. Zhang, "Boolean Satisfiability: From Theoretical Hardness to Practical Success. *Review Article: Communications of the ACM,* Vol. 52(8), Aug. 2009

[6] H. M. Ali, S. Ashrafinia, J.C. Liu and D. C. Lee, "Broadband Wireless Network Planning Using Evolutionary Algorithms," *Proceedings of IEEE Congress on Evolutionary Computation (CEC)* Cancún, México, June 20-23, 2013

[7] H. M. Ali, J. S. Oberoi, JC Liu and D. Lee, "Base Station and Relay Station Broadband Network Planning Using Immune Quantum Evolutionary Algorithm" *Proceedings of IEEE vehicular technology conference (VTC-Fall),* Las Vegas, USA, Sep. 2-5, 2013

[8] S. Ashrafinia, U. Pareek, M. Naeem, and D. Lee, "Binary Artificial Bee Colony for Cooperative Relay Communication in Cognitive Radio Systems," *Proceedings of IEEE International Conference on Communications* (ICC), Ottawa, Canada, Jun 10-15, 2012

[9] J.S. Oberoi, U. Pareek, M. Naeem and D.C. Lee, "EDA-based joint power, subcarrier allocation and relay assignment scheme for multiuser relaying in OFDMA-based cognitive radio systems," *Proceedings of IEEE 5th International Conference on Signal Processing and Communication Systems (ICSPCS)* Honolulu, Hawaii, Dec. 12-14, 2011

[10] T. D. Seeley, The wisdom of the hive: the social physiology of honey bee colonies, Cambridge, MA: Harvard University Press.

[11] V. Tereshko, Parallel Problem Solving from Nature PPSN VI: Reaction-diffusion model of a honeybee colony's foraging behavior, Lecture Notes in Computer Science, Springer – Berlin

[12] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numericalfunction optimization: artificial bee colony (ABC) algorithm", *Journal of Global Optimization*, Vol. 39(3), Apr 13, 2007.

[13] H. M. Ali, S. Ashrafinia, J. C. Liu and D. C. Lee, "Wireless Mesh Network Planning Using Quantum Inspired Evolutionary Algorithm" *Proceedings of IEEE Vehicular Technology Conference (VTC-Fall), San Francisco*, USA, Sep. 2011

[14] X. Liu, Y. Wu, and J. Ye, "An Improved Estimation of Distribution Algorithm in Dynamic Environments," *Proceedings of Fourth International Conference on Natural Computation, (ICNC '08),* Jinan, China, Oct.18-20, 2008

[15] http://pic.dhe.ibm.com/infocenter/spssstat/ (Accessed on January 27, 2014).

[16] B. Shabash, K. C. Wiese, "pEvoSAT: A Novel Permutation Based Genetic Algorithm for Solving the Boolean Satisfiability Problem" *Proceedings Genetic and Evolutionary Computation Conference (GECCO'13), Amsterdam, The Ntherlands. July 6-10, 2013*