Int J Syst Assur Eng Manag (August 2018) 9(4):741–754
https://doi.org/10.1007/s13198-017-0620-x

ORIGINAL ARTICLE

Harmony search based memetic algorithms for solving sudoku

Assif Assad[1] · **Kusum Deep**[1]

Received: 8 February 2017 / Published online: 27 April 2017

Abstract The development of hybrid procedures for optimization focuses on enhancing the strength and compensating for the weakness of two or more complementary approaches. The goal is to intelligently combine the key elements of the competing methodologies to create a superior solution procedure. The objective of this paper is to explore the hybridization between Harmony Search and Hill Climbing algorithm by utilizing the exploration power of the former and exploitation power of the latter in the context of solving Sudoku which is a well-known hard combinatorial optimization problem. We call this hybrid algorithm Harmony Search Hill Climber (HSHC). In order to extend the exploration capabilities of HSHC it is further modified to create three different algorithms namely Retrievable Harmony Search Hill Climber (RHSHC), Global Best Retrievable Harmony Search Hill Climber (GB-RHSHC) and Random Best Retrievable Harmony Search Hill Climber (RB-RHSHC). Comparing the four algorithms proposed in this paper RHSHC outperforms its three variations in terms of effectiveness. Experimental results demonstrate that RHSHC perform significantly better than standard Harmony Search algorithm and standard Hill climber algorithm. On comparing RHSHC with the genetic algorithm it has been concluded that former outperforms latter both in terms of effectiveness and efficiency particularly for Hard and Expert level puzzles. Comparing RHSHC and hybrid AC3-tabu search algorithm it has been concluded that RHSHC is very competent to hybrid AC3-tabu search algorithm.

Keywords Harmony search · Hill climbing · Sudoku · Memetic algorithm · Evolutionary algorithm

✉ Assif Assad
assifassad@gmail.com

Kusum Deep
kusumdeep@gmail.com

[1] Department of Mathematics, Indian Institute of Technology Roorkee, Roorkee 247667, India

1 Introduction

SUDOKU is a Japanese puzzle which consists of an N × N square that is divided into $\sqrt{N}$ sub-squares, each of size $\sqrt{N} \times \sqrt{N}$. Here N is a perfect square and is known as the order of the Sudoku. In the beginning, there are some static numbers (called givens or fixed) in the puzzle. The game is to fill all non givens such that each row, column and sub-square contains each integer from 1 to N exactly once. Sudoku is a well-known NP complete problem (Takayuki and Takahiro 2003). The difficulty level of the Sudoku puzzle is determined by around twenty factors and the number of initial givens has no or little role in it (Mantere and Koljonen 2006). Figure 1 is an example of a Sudoku puzzle of order 9 and Fig. 2 represents its solution. In the solution, each row, column and sub-square contains integers from 1 to 9 exactly once, further the givens remain intact. Sudoku is linked to real world applications including conflict free wavelength routing in wide band optical networks, statistical design and error correcting codes, as well as timetabling and experimental design (Jones et al. 2008). Another closely related problem to sudoku is generating threshold matrix for halftoning grayscale images (Mantere and Koljonen 2006).

Harmony Search (HS) is a musicians behavior inspired evolutionary algorithm developed in 2001 by Geem et al.

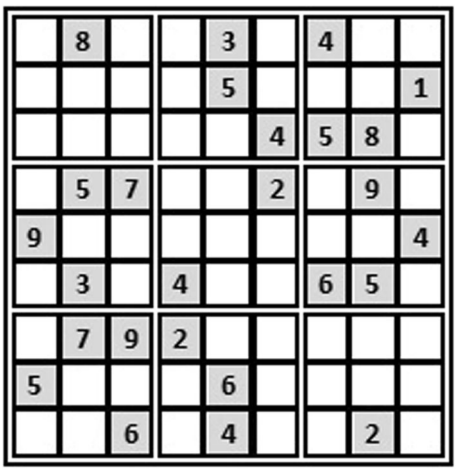

Fig. 1 A sudoku puzzle with 26 givens

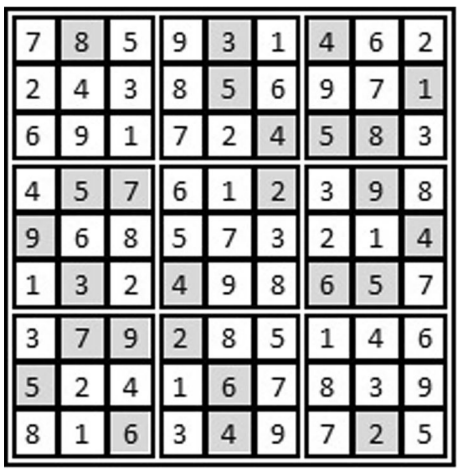

Fig. 2 Solution of the sudoku puzzle given in Fig. 1

(2001), though it is a relatively new meta heuristic algorithm, its effectiveness and advantages have been demonstrated in various applications like structural design (Gholizadeh and Barzegar 2013), load dispatch problem in electrical engineering (Wang and Li 2013), multi objective optimization (Nekooei et al. 2013), rostering problems (Hadwan et al. 2013), classification and feature selection (Diao and Shen 2012; Fattahi et al. 2015). HS has demonstrated several advantages like simplicity, flexibility, adaptability, generality,and scalability over traditional optimization techniques (Al-Betar et al. 2012) and has been particularly successful on combinatorial optimization problems where it has outperformed other meta heuristic algorithms like genetic algorithm as well as the traditional branch and bound method (Geem 2005). HS works by generating a new vector that encodes a candidate solution, after considering the selection of all existing quality vectors. This forms a contrast with conventional evolutionary approaches such as GAs that consider only two (parent) vectors in order to produce a new (child) vector. It increases the exploration capabilities of HS (Diao and Shen 2012) and hence has been preferred over other global search techniques like GA in this article.

Hill climbing is a local search optimization operator. It is an iterative algorithm that starts with an arbitrary solution of a problem, then explores to find a better solution by incrementally changing a single component of the solution. If the change produces a better solution, the new solution is accepted, repeating until no further improvements can be found.

Memetic Algorithms (MA) are a class of stochastic global search heuristics in which population based Evolutionary algorithms are hybridized with problem specific solvers, typically local search heuristic techniques to improve the quality of the solution (Ong et al. 2006). The name is inspired by Richard Dawkińs concept of *meme*, which represents a unit of cultural evolution that can represent local refinement (Dawkins 2006). MAs have arisen as a reciprocation to the problem encountered in the conventional evolutionary algorithms which are good at global exploration of the search space however can take relatively large time to find the optimal with sufficient precision (Ong et al. 2006). This often limits the practicality of evolutionary algorithms in many real world problems where computational time is of paramount importance. This hybridization between global and local search methods refereed to as MA has been shown to be more efficient (i.e., requiring orders of magnitude fewer evaluations to converge) and effective (i.e., identifying high quality solutions that would otherwise be unreachable by evolutionary algorithm or local search alone) than traditional evolutionary algorithms on several problem domains (Al-Betar et al. 2012). MA are successful and popular for solving optimization problems in many contexts (Al-Betar et al. 2012; Ishibuchi et al. 2003; Chan et al. 2012; Sharma et al. 2016; Jadon et al. 2015; Sharma et al. 2013). Hart et al. (2005) gives an elaborate review of MAs.

Many attempts have been made in literature to solve Sudoku puzzles these include exact search methods such as constraint programming (Moon and Gunther 2006) and Boolean satisfiability (Lynce and Ouaknine 2006) to heuristic and metaheuristic based algorithms including Simulated Annealing (SA) (Lewis 2007), GA (Mantere and Koljonen 2006), Ant Systems (Mullaney 2006), Differential Evolution (DE) (Boryczka and Juszczuk 2012) to name a few. Other less traditional techniques in this context such as Sinkhorn balancig (Moon et al. 2009), rewriting rules (Moon et al. 2009) and entropy minimization (Gunther and Moon 2012) has also been proposed to tackle this problem.

The objective of this paper is to create an algorithm namely Harmony Search Hill Climber (HSHC) by

hybridizing HS and Hill Climbing operator with an aim to solve Sudoku. In order to increase the exploration capabilities of HSHC, three variations of HSHC have been proposed. Since the proposed algorithms use hill climbing operator they can be termed to fall in the category of memetic algorithms.

The rest of the paper is structured as follows. Section 2 is an introduction to HS algorithm, a review of literature on Sudoku puzzle is provided in Sect. 3. Section 4 presents the proposed HSHC algorithm along with its three variations. In Sect. 5 the numerical results are discussed and analyzed and in Sect. 6 RHSHC is compared with other heuristic algorithms. Finally the conclusions are drawn in Sect. 7.

2 Harmony search

In order to explain the Harmony Search in detail, let us first idealize the improvisation process by a skilled musician. When a musician is improvising there are three possible choices:

(1) Play any famous piece of music exactly from his memory.
(2) Play something similar to a known piece (thus adjusting the pitch slightly).
(3) Compose new or random notes.

Geem et al. (2001) formalized these three options into quantitative optimization process in 2001, and the three corresponding components become usage of harmony memory (HM), pitch adjusting, and randomization. The usage of harmony memory is similar to the choice of the best-fit individuals in genetic algorithms (GA). In order to use this memory effectively, it is typically assigned a parameter called harmony memory consideration rate (HMCR $\in$ [0, 1]). If this rate is too low (near 0), only few best harmonies are selected and thus convergence of algorithm may be too slow. If this rate is very high (near 1), it results in exploitation of the harmonies in the HM, thus other harmonies are not explored well, leading to potentially wrong solutions. Typically HMCR $\in$ [.7, .95] is used. The second component is pitch adjustment determined by pitch bandwidth (BW) and pitch adjustment rate (PAR) though in music pitch adjustment means to change the frequencies, it corresponds to generating a slightly different solution in the Harmony Search algorithm. Pitch can be adjusted linearly or nonlinearly however most often linear adjustment is used. So we have

$$H^i_{new} = H^i_{old} + BW \times r_i \quad \text{where} \quad r_i \in [-1, 1] \quad \text{and} \quad 1 \le i \le N \tag{1}$$

where H^i_{old} is the *i th* component of the existing harmony or solution and H^i_{new} is the *i th* component of new harmony after the pitch adjusting action. The Eq. (1) essentially produces a new solution around the existing solution by varying the solution slightly by a small random amount. Here r_i is a random number generated in the range of $[-1, 1]$ and N is total number of components in the harmony. The pitch adjusting rate (PAR) controls the degree of adjustment. A low pitch adjusting rate with a narrow bandwidth can slow down the convergence of HS because of limitation in exploration of only a small subspace of the whole search space. On the other hand an extremely high PAR with a wide bandwidth may cause the solution to swing around some potential optimal solution. Thus most commonly used value of PAR $\in$ [.1, .5]. The third component is the randomization, which is to increase the exploration of the search space. Although pitch adjustment has a similar role, but it is limited to close neighborhood of harmony thus corresponds to local search. The use of randomization can push the system further to explore various diverse solutions so as to find the global optima. The pseudo code of harmony search is shown as Algorithm 1. It is evident from the pseudo code that the probability of randomization is 1-HMCR and the probability of pitch adjustment is HMCR $\times$ PAR. In the pseudo code H represents a potential solution or harmony, rand $\in$ [0, 1] is a uniformly distributed random number generator and HMS is the size of harmony memory.

Algorithm 1 HARMONY SEARCH ALGORITHM (HSA)

```
Define Objective function f(H).
Define harmony memory consideration rate (HMCR).
Define pitch adjustment rate (PAR) and bandwidth(BW).
Define Harmony Memory Size (HMS).
Initialize Harmony Memory (HM).
while (Stopping Criteria Not Reached) do
  Find current WORST and BEST harmony in HM.
  for i = 1 to N do
    if (rand ≤ HMCR) then
      H[i] = HM[j][i] where j ∈ (1,2, ... ,HMS)
      if (rand ≤ PAR) then
        H[i] = H[i]± rand × BW
      end if
    else
      Generate H[i] using randomization.
    end if
    if (H is better than worst Harmony in HM) then
      Update HM by replacing WORST harmony by H.
    end if
  end for
end while
print Best Harmony as obtained solution.
```

3 Related work

Deterministic algorithms based on branch and bound strategy have been used to solve Sudoku, since Sudoku is an NP-complete problem, thus we cannot find a polynomial time algorithm for all possible problem instances, unless P = NP (Garey and Johnson 1979). Thus researchers have made efforts to solve Sudoku puzzles using various meta heuristic algorithms.

For instance Mantere and Koljonen have used Genetic Algorithm (GA) to solve Sudoku puzzle in Mantere and Koljonen (2006). The algorithm is extension of the one devoted to solve magic square problems. In the algorithm each chromosome is represented as an integer array with size 81. Each chunk of 9 digits starting from left corrosponds to 3 × 3 sub block of Sudoku. Each sub block is initialized with numbers from 1 to 9 such that there is no repetition of digits and givens remain intact. The crossover site is only at the boundary of sub blocks, the mutation used is: swap mutation, 3-swap mutation and insertion mutation and is allowed only with in the sub block. The algorithm has been tested on different categories of Sudoku puzzles although good performance have been exhibited in solving easy and medium type Sudoku however the success rate for challenging, difficult and super difficult puzzles is 30, 4 and 6% respectively. Das et al. (2012) have proposed Retrievable GA algorithm by modifying the GA algorithm proposed in Mantere and Koljonen (2006). The main difference between Retrievable GA and the one proposed in Mantere and Koljonen (2006) is that in former population is re initialized after certain number of generations (which depend on difficulty level of puzzle). Experimental results demonstrate that Retrievable GA performs better than the one proposed in Mantere and Koljonen (2006) both in terms of effectiveness and efficiency (Das et al. 2012). The success rate shown by Retrievable GA in solving easy and medium puzzles is 100% and for difficult and superdifficult puzzles it is 16 and 9% respectively. Nicolau and Ryan (2006) have used Genetic algorithm using Grammatical Evolution (GAuGE) for solving sudoku, GAuGE uses position independent representation. Each phenotype variable is encoded as a genotype string along with an associated phenotype position to learn linear relationships between variables. The GAuGE algorithm has shown promising results for majority of test puzzles, however as reported in Nicolau and Ryan (2006) there were some test puzzles on which the algorithm failed completely. Li and Deng (2011) have modified the various important operators of GA in a bold way so that the algorithm has higher reliability, quicker convergence and better stability. Sato and Inoue (2010) have proposed a hybrid GA local search algorithm to tackle the sudoku puzzles and have shown good performance particularly for solving difficult puzzles. In Deng and Li (2013) a hybrid GA has been utilized to solve Sudoku. The proposed algorithm was able to solve majority of easy puzzles however the success rate for difficult and superdifficult was 17 and 0% respectively. In order to accelerate the speed of Genetic algorithms for sudoku solving a parallel GA has been proposed in Sato et al. (2013).

Hill Climbers have been tested to solve sudoku puzzle in Moraglio et al. (2006). In order to restrict the search space explored by Hill Climber the concept of Smart Square Mutation has been applied. This mutation applies the most obvious constraint to the possible values Sudoku can take. For example if a row has a fixed '9' then '9' is removed from the set of possible values of that row, the same concept is extended to columns and sub squares. Though the proposed Hill climbing algorithm powered by Smart Square Mutation succeeded in solving easy type puzzles however it completely failed in solving medium and hard ones.

In Lewis (2007) a simulating annealing algorithm has been presented to solve sudoku, however the approach is mainly centered on creating solvable problems than solving hard Sudoku puzzles.

A hybrid tabu search algorithm has been proposed to solve the Sudoku puzzle in Soto et al. (2015) and the algorithm has shown very promising results for solving difficult and superdifficult puzzles.

In Boryczka and Juszczuk (2012) Differential Evolution (DE) has been tested on sudoku puzzles, the authors have further tested the algorithm on classifying the Sudoku puzzles depending on their difficulty level. Though encouraging results have been reported however the algorithm has been tested on only few puzzles.

An evolutionary algorithm employing filtered mutations have been proposed in Wang et al. (2015) to tackle Sudoku puzzle, the algorithm has been tested on 6 puzzles (2 each of type easy, medium, difficult and super difficult) and has shown good results particularly in solving difficult and super difficult puzzles.

A hybrid AC3-tabu search algorithm for solving sudoku has been proposed in Soto et al. (2013). The algorithm has been created by combining tabu search with an arc-consistency 3 (AC3) algorithm that acts as a domain reducer. This integration reduces the number of tabu search iterations thus increases the convergence speed of the algorithm. As illustrated in Simonis (2005) Sudoku can be represented as constraint network, thus consequence

techniques from constraint satisfaction can be applied on them. Arc-consistency is one of the most utilized filtration technique in constraint satisfaction for reducing the search space of combinatorial problems. Arc-consistency is formally defined as local consistency with in the constraint programming field (Rossi et al. 2006). A local consistency defines properties that the constraint problem must satisfy after constraint propagation. The hybrid AC3-tabu search algorithm has shown excellent performance on all types of Sudoku puzzles and has been compared with genetic algorithm proposed in Manter and Koljonen (2007). The simulation results show that former is significantly effective than later particularly on hard Sudoku puzzles.

In Soto et al. (2014) a Cuckoo Search Algorithm with Geometric Operators has been utilized for solving Sudoku Problems. The algorithm was able to solve easy and medium sudoku with approximately 100% success rate and hard puzzles with approximately 65% success rate in 10,000 iterations and if allowed to run for unlimited iterations the success rate for all types of puzzles was 100%.

HS algorithm has been used to solve Sudoku puzzle in Geem (2007) however there are three main limitations in that article.

(1) In order to carry out experimentation only two Sudoku puzzles one of type Easy another of type Hard has been used to conclude that the algorithm solves easy problem very efficiently and fails to solve the hard problem, however in randomized algorithms one can't jump on conclusion by taking such a small example set.
(2) There are different types of Sudoku puzzles like Beginner, Easy, Medium, Hard, and Expert however only Easy and Hard problem has been attempted.
(3) The fitness function used is

$$Minimize \quad Z = \sum_{i=1}^{9}\left|\sum_{j=1}^{9} x_{ij} - 45\right| + \sum_{j=1}^{9}\left|\sum_{i=1}^{9} x_{ij} - 45\right| + \sum_{k=1}^{9}\left|\sum_{(l,m)\in B_k} x_{lm} - 45\right| \tag{2}$$

where $X_{ij} \in \{1, 2, \ldots, 9\}$is the (i, j)th element of Sudoku

The first term in Eq. (2) represents the penalty function for each horizontal row; the second term for each vertical column; and the third term for each block. The solution is reached when there is no violation (i.e. repeating number) in rows, columns and blocks thus for solution Eq. (2) evaluates to Zero. There are two main disadvantages with this fitness function (Eq. 2).

(1) Even if the fitness function evaluates to zero it is not guaranteed that the solution has been found. A detailed discussion on the limitations of the above mentioned fitness function (Eq. 2) can be found in Weyland (2015).
(2) It gives more penalty to repetition of higher face value digits than lower face value digits. Let there be two solutions A and B such that in solution A, the digit '9' occurs twice in some row and in solution B, the digit '1' occurs thrice in some row. Then according to the fitness function (Eq. 2) solution A is better than solution B, although solution A has more violation than solution B.

Thus there is a scope to modify the basic HS algorithm so that it can be effective on all categories of Sudoku puzzles viz. Beginner, Easy, Medium, Hard and Expert.

4 Proposed HSHC algorithm

In this paper a hybrid algorithm of Harmony search and Hill Climbing has been proposed, namely Harmony Search Hill Climber (HSHC). In order to increase the exploration potentiality of HSHC it has been modified to create another algorithm namely Retrievable Harmony Search Hill Climber (RHSHC), further two variations of RHSHC namely Global Best Retrievable Harmony Search Hill Climber (GB-RHSHC) and Random Best Retrievable Harmony Search Hill Climber (RB-RHSHC) have been proposed.

Algorithm 2 is the Pseudo code of HSHC algorithm. In Algorithm 2 the parameters HMS, HMCR, NH_SIZE respectively denote Harmony Memory size, Harmony Memory Consideration Rate and Size of neighborhood to be explored by Hill Climbing operator.

Algorithm 2 Harmony Search Hill Climber (HSHC)

1: Initialize Algorithm parameters HMS, HMCR, NH_SIZE.
2: Initialize Harmony Memory.
3: Evaluate each harmony of harmony memory using fitness function (equation 4).
4: **while** Stoping Condition Not Reached **do**
5: Find BEST and WORST Harmony in HM.
6: **if** rand$(0, 1)$ is less than HMCR **then**
7: With probability P select randomly one of the Harmony from HM otherwise generate a new harmony using the harmonies in HM, name it ROOT.
8: **else**
9: Generate a Harmony Randomly, name it ROOT.
10: **end if**
11: Evaluate ROOT using fitness function (equation 4).
12: **if** ROOT is solution **then**
13: STOP.
14: **end if**
15: **for** $i = 1$ TO NH_SIZE **do**
16: Select randomly two cells of ROOT with different values, swap their contents to obtain a neighbor of ROOT, name it NBR.
17: **if** NBR is Solution **then**
18: STOP.
19: **end if**
20: **if** NBR is better than or marginally inferior to BEST **then**
21: set ROOT=NBR.
22: **end if**
23: **if** NBR is better than worst Harmony **then**
24: update Harmony Memory by replacing WORST Harmony by NBR.
25: **end if**
26: **end for**
27: **end while**

Detailed description of the HSHC pseudo code (Algorithm 2)

The working of Algorithm 2 is described in the following steps.

Step 1

In this step the free parameters of algorithm like Harmony Memory Size (HMS), Harmony Memory Consideration Rate (HMCR) and Neighborhood Size (NH_SIZE) to be explored by Hill Climber operator are initialized.

Step 2

Initialization of harmony memory (HM) is performed in this step. So far as the structure of the harmony is concerned, each harmony represents a complete Sudoku. Thus harmony is represented by a vector of dimension N × N where N is the order of the Sudoku puzzle. Harmony Memory (HM) is an array of such vectors with dimension HMS. Mathematically the structure of HM is

$$H_1 = \begin{bmatrix} h_{11}^1 & h_{12}^1 \ldots h_{1N}^1 \\ h_{21}^1 & h_{22}^1 \ldots h_{2N}^1 \\ \ldots\ldots\ldots\ldots\ldots \\ h_{N1}^1 & h_{N2}^1 \ldots h_{NN}^1 \end{bmatrix}$$

$$H_2 = \begin{bmatrix} h_{11}^2 & h_{12}^2 \ldots h_{1N}^2 \\ h_{21}^2 & h_{22}^2 \ldots h_{2N}^2 \\ \ldots\ldots\ldots\ldots\ldots \\ h_{N1}^2 & h_{N2}^2 \ldots h_{NN}^2 \end{bmatrix}$$

$$\vdots$$

$$H_{HMS} = \begin{bmatrix} h_{11}^{HMS} & h_{12}^{HMS} \ldots h_{1N}^{HMS} \\ h_{21}^{HMS} & h_{22}^{HMS} \ldots h_{2N}^{HMS} \\ \ldots\ldots\ldots\ldots\ldots \\ h_{N1}^{HMS} & h_{N2}^{HMS} \ldots h_{NN}^{HMS} \end{bmatrix}$$

where H_n is the *nth* harmony, $f(H_n)$ is the fitness value of the *nth* harmony and h_{ij}^n is cell at row i and column j in the *n*th harmony of HM. While initializing the harmonies a constraint is defined such that each row in the harmony contain numbers from 1 to N exactly once without repetition. Mathematically (*i*, *j*)th element of harmony n denoted by h_{ij}^n must satisfy following constraints

$$\begin{aligned} &\textit{for each ith row}, (1 \le i \le N), h_{ij}^n \ne h_{ik}^n, \\ &\quad k \ne j, 1 \le j, k \le N, n \in \{1, 2, \ldots, HMS\} \end{aligned} \tag{3}$$

If all the elements in a harmony satisfy above constraints (Eq. 3) it is guaranteed that the frequency of occurrence of each number in the harmony is N. This is an important achievement because in a valid Sudoku solution frequency of occurrence of each number must be exactly N. Line number 15 through 27 of Algorithm 2 constitute the hill climbing operator, where neighbors of a given harmony are generated by exchanging cell values and this operator will never converge to solution if the above mentioned constraint on frequency of occurrence of digits is not obeyed by harmony. Therefore during random initialization of HM it is made sure that each row in the harmony must obey the constraint represented by Eq. (3). Though there are many ways to achieve it, let us consider following two ways.

Consider the pseudo code given as Algorithm 3. In Algorithm 3 line number 3* generates random number from 1 to N and assigns it to r, line number 5* compare already assigned numbers in row i of H with r if any of the numbers happen to be same as that of r, r gets regenerated otherwise

H[i][j] is assigned r where H is the harmony with dimension N × N and H[i][j] is the $(i, j)th$ element of H . It is easy to verify that the worst case time complexity of Algorithm 3 is greater than $O(N^3)$ where N is the order of the Sudoku.

Another algorithm to obtain randomization in an array is called Richard Durstenfeld algorithm (Durstenfeld 1964). Pseudo code of the algorithm for random initialization of Sudoku using Richard Durstenfeld algorithm is given as Algorithm 4. In Algorithm 4 line number 2* and 3* sequentially assigns numbers from 1 to N to row i of H, then in line number 6* and 7* one of the numbers is randomly picked and shifted to end, the loop in line number 5* repeats this process for all the N numbers. Since the complexity of Algorithm 4 is $O(N^2)$ so it is considered for random initialization of harmony in HM. Further it must be noted during initialization no special care is provided to givens/fixed cells, rather they are also initialized randomly and for any violation of givens the harmony will be penalized by fitness function.

Algorithm 3 Random Initialization of Sudoku

```
1*  for i=1 to N do
2*    for j=1 to N do
3*      r=rand(1,N)
4*      for l=1 to j-1 do
5*        if H[i][j]=r then
6*          GOTO Line 3*
7*          H[i][j]=r
8*        end if
9*      end for
10*   end for
11* end for
```

Algorithm 4 Richard Durstenfeld Algorithm for Random Initialization of Sudoku

```
1* for i=1 to N do
2*   for j=1 to N do
3*     H[i][j]=j
4*   end for
5*   for k=N down to 1 do
6*     r=rand(1,k)
7*     swap H[i][r] and H[i][k]
8*   end for
9* end for
```

Step 3

In this step each Harmony is evaluated to determine its fitness. The limitation of fitness function used in Geem (2007) has already been discussed in Sect. 3, so fitness function proposed by Das et al. (2012) has been used with slight modification. The fitness function proposed in this paper consists of four parts, namely row-fitness, column-fitness, sub-square-fitness and givens violation fitness. The first three fitness terms have equal weightage, however the last one has W times more weightage than first three terms. Thus the overall fitness function becomes

$$\begin{aligned} \textit{Fitness function} = \textit{Row fitness} + \textit{Column fitness} \\ + \textit{Subsquare fitness} - W \times \textit{Givens violation} \end{aligned} \tag{4}$$

The fitness function (Eq. 4) attains the maximum possible value when the solution is reached. Each row entry is compared with all the remaining entries to its right. If the two entries are not equal, row fitness value is incremented by one otherwise it remains same. Thus in the solution the contribution from each row is $N(N-1)/2$ (sum of first N–1 natural numbers). Hence for N rows it is $N^2(N-1)/2$. Similar results hold for columns and sub squares. Hence for an $N \times N$ Sudoku puzzle the maximum possible fitness value is $3N^3(N-1)/2$. It must be noted that the contribution of penalty for violating givens will be zero in the solution. The fitness value for the solution of a 9 × 9 Sudoku puzzle is 972. Thus when the fitness function attains its maximum possible value it is guaranteed that solution has been obtained. Expressing this concept in mathematical form

$$f(i,j,k,l) = \begin{cases} 0 & \text{if} \quad (\mathrm{i},\mathrm{j}) = (\mathrm{k},\mathrm{l}) \\ 1 & \textit{otherwise} \end{cases}$$

where (i,j) and (k,l) refer to two entries of N × N Sudoku puzzle. The fitness function for each row is defined as

$$\textit{Row fitness} = \sum_{i=1}^{N} \sum_{j=1}^{N-1} \sum_{l=j+1}^{N} f(i,j,i,l) \tag{5}$$

The fitness function for each column is defined as

$$\textit{Column fitness} = \sum_{j=1}^{N} \sum_{i=1}^{N-1} \sum_{l=i+1}^{N} f(i,j,l,j) \tag{6}$$

The fitness function for each sub square is defined as

$$\begin{aligned} &\textit{Sub Square fitness} \\ &= \sum_{i=1}^{N} \left[\sum_{q=1}^{\sqrt{N}} \left\{ \sum_{j=1+(q-1)\sqrt{N}}^{q\sqrt{N}-1} \sum_{l=j+1}^{q\sqrt{N}} f(i,j,i,l) + \sum_{\substack{r=1+(q-1)\sqrt{N} \\ i \neq t\sqrt{N}, t \in Z^+}}^{i+\sqrt{N}-i(mod\sqrt{N})} \sum_{k=i+1}^{q\sqrt{N}} \sum_{s=1+(q-1)\sqrt{N}}^{q\sqrt{N}} f(i,r,k,s) \right\} \right] \end{aligned} \tag{7}$$

The fitness function for violating fixed/givens is defined as

$$Givens\ violation = \sum_{i=1}^{N}\sum_{j=1}^{N} l(i,j)$$
$$where \quad l(i,j) = \begin{cases} 1 & \text{if flag}(i,j) \neq 0 \text{ and } (i,j) \neq \text{flag}(i,j)) \\ 0 & otherwise \end{cases} \tag{8}$$

where flag is an N × N matrix whose $(i, j)th$ element is 0 if $(i, j)th$ element is not given otherwise it is equal to given/fixed $(i, j)th$ element and Z^+ is the set of all positive integers.

Hence from Eq. (4) the fitness function is the sum of Eqs. (5), (6), (7) minus W times Eq. (8). After thorough experimentation the value for W in Eq. 4 was fixed as 8.

Step 7

This step is executed with probability HMCR and in this step a new harmony say H^{new} is produced either by selecting an existing harmony from HM with probability P or by combing the harmonies in HM with probability 1-P. While producing a new harmony using existing harmonies a simple mechanism is used. While generating $i\ th$ $(1 \leq i \leq N)$ row of H^{new}, one of the harmonies from HM is randomly selected and its $i\ th$ row is added to H^{new}. The above procedure is repeated for all the N rows of H^{new}. The optimum value of P was found out to be .95 after thorough experimentation.

Step 9

The aim of this step is to speedup convergence by incorporating problem specific knowledge in harmony creation. This step must be executed with very low probability because it increases the probability of being stuck in local optimal. In this step a harmony is randomly generated however during random harmony creation two facts are kept in mind one the givens must remain intact another there must be no repetition of numbers in rows and columns. However repetition of numbers in blocks is allowed. Mathematically h_{ij} the $(i, j)th$ element of H^{new} must satisfy following constraints if it is not fixed.

$$for\ each\ element\ \ h_{ij},\ h_{ij} \neq h_{ik}\ (j \neq k)\ and$$
$$h_{ij} \neq h_{lj}\ (l \neq i), (i,j,k,l) \in \{1,2\ldots,N\} \tag{9}$$

Steps 15 through 27 of Algorithm 2 constitute the Hill Climbing operator where a neighbor of a Harmony is generated by exchanging contents of two cells having different values.

Step 20

In this step NBR is compared with the best Harmony and in case NBR is better than or slightly inferior than BEST, NBR becomes ROOT. The reason for this move is that since NBR seems to be very promising as it can compete with the best Harmony it is reasonable to concentrate on this new harmony by exploring it further. If the difference between the fitness value of BEST and NBR is less than 3 it is considered to be marginally inferior to BEST and thus eligible to be explored further. After through experimentation the optimal value of T was found out to be 3.

Steps 23 and 24

If any harmony say NBR produced during hill climbing is better than the WORST Harmony in Harmony Memory, the WORST harmony in HM is replaced by that harmony.

One of the main disadvantages of evolutionary algorithms (including HS) is premature convergence due to lack of diversity in population, so in order to overcome this issue the concept of catastrophic mutation form GA (Jin and Li 1997) has been adopted in HSHC and three new algorithms namely RHSHC, GB-RHSHC and RB-RHSHC proposed. In Catastrophic mutation the population is unsettled by very high mutation rate (typically greater than 0.8) so as to recover the population diversity and thus avoid premature convergence in GA. The RHSHC is obtained by applying catastrophic mutation to HSHC algorithm. If the BEST Harmony in HSHC does not change 150,000 function evaluations it is assumed that the algorithm has got stuck in some local optimal and thus Harmony memory is reinitialized.

The difference between GB-RHSHC and RHSHC is that in GB-RHSHC when the reinitialization of HM is performed the BEST Harmony in the Harmony Memory is copied as such and the other HMS-1 harmonies are randomly initialized, however in RHSHC the entire Harmony memory is randomly initialized.

The RB-RHSHC is also obtained by slightly modifying RHSHC algorithm. In RB-RHSHC when the reinitialization of HM is performed one of the Harmony from Harmony Memory (not necessarily the BEST) is copied as such and the other HMS-1 harmonies are randomly initialized.

The time complexity of fitness function (Eq. 4) is $O(N^3)$ and hence the overall time complexity of HSHC is $O(KN^3)$ where N is the order of sudoku and K is the allowed number of fitness function evaluations. It should be noted that the time complexity of RHSHC, GB-RHSHC, RB-RHSHC remains same as that of HSHC.

5 Computational experiment

In order to check the effectiveness and efficiency of the proposed algorithms a set of 25 Sudoku puzzles (of order 9) five each of type Beginner, Easy, Medium, Hard and Expert have been taken from the web site www.sudoku.com and each problem has been tested 30 times. Determining the optimal setting for free parameters in a meta

heuristic algorithm is a hyper optimization problem, however after thorough experimentation following parameter setting was found out to be effective for most if not all the cases: HMS = 40, HMCR = 0.99 and NH_SIZE = 120. The stopping criteria for a run is either the solution is found or maximum execution time of 35 s is attained. The experimentation was carried out on a laptop having specification- Intel CORE i3 processor, 4 GB of RAM and Windows 8.1 Operating System.

Tables 1, 2, 3, 4 demonstrate the performance of the proposed algorithms HSHC, RHSHC, GB-RHSHC and RB-RHSHC respectively. The columns of all the four tables (1, 2, 3, 4) from left to right represent: Puzzle type (PUZZLE TYPE), Percentage of runs that are able to find solution of a given puzzle (SP), Minimum execution time (MINT), Maximum execution time (MAXT), Average execution time (AVGT), Standard deviation of execution time (SDT), Minimum number of fitness Function Evaluations performed (MINFE), Maximum number of fitness Function Evaluations performed (MAXFE), Average number of fitness Function Evaluations performed(AVGFE) and Standard Deviation of number of fitness Function Evaluations performed (SDFE).The above statistics in terms of execution time and number of fitness function evaluations required have been obtained for successful runs only.

Figure 3 compare the performance of proposed four algorithms in terms of success rate on different types of Sudoku puzzles. Figures 4, 5 and 6 respectively compare the performance of the four algorithms in terms of Minimum, Maximum and Average execution time required for successful runs. Figures 7, 8 and 9 respectively compare the performance of the four algorithms in terms of Minimum, Maximum and Average number fitness function

Table 1 HSHC performance

Puzzle type	SP	MINT	MAXT	AVGT	SDT	MINFE	MAXFE	AVGFE	SDFE
Beginner	70.67	0.251	32.911	4.892	2.734	40041	4979069	703649.7	445102.9
Easy	66	0.436	28.805	4.221	2.847	67853	4645532	657033.3	433648.7
Medium	33.33	0.796	32.674	5.073	2.022	126055	4803732	792300.5	290206.2
Hard	33.33	0.719	29.014	6.895	2.399	116955	4759383	1114436	392600.2
Expert	30.67	0.945	28.009	5.804	2.921	144262	3647316	830308.3	355427.8

Table 2 RHSHC performance

Puzzle type	SP	MINT	MAXT	AVGT	SDT	MINFE	MAXFE	AVGFE	SDFE
Beginner	100.00	0.260	29.121	3.103	2.301	41435	4358130	672271.0	635898.0
Easy	100.00	0.340	32.653	6.034	2.575	53750	4751663	634465.2	608455.6
Medium	93.33	0.691	31.370	10.196	2.582	107590	4700497	1926207.5	1152346.3
Hard	88.67	1.029	31.267	12.771	1.798	157017	4954494	2025149.5	303393.9
Expert	80.00	1.025	33.828	13.579	1.301	159260	4868901	1911467.8	252190.1

Table 3 GB-RHSHC performance

Puzzle type	SP	MINT	MAXT	AVGT	SDT	MINFE	MAXFE	AVGFE	SDFE
Beginner	81.33	0.296	22.399	3.289	3.186	47244	3581089	505804.7	477320.1
Easy	66.00	0.396	34.037	4.945	4.196	58969	4858880	766863.1	622109.7
Medium	57.33	0.750	33.158	7.861	5.297	121566	4857597	1196316.0	772401.0
Hard	49.33	0.780	28.082	8.222	1.739	109090	4170662	1237255.9	248678.5
Expert	42.67	0.846	34.154	11.091	2.914	120754	4975036	1678574.1	450827.2

Table 4 RB-RHSHC performance

Puzzle type	SP	MINT	MAXT	AVGT	SDT	MINFE	MAXFE	AVGFE	SDFE
Beginner	84.00	0.287	32.235	3.052	1.402	45830	4000409	665515.9	415526.8
Easy	76.67	0.331	26.951	5.445	1.580	51823	4522048	828088.2	695742.7
Medium	72.67	0.682	31.621	8.002	2.614	106805	5767232	1345171.3	538997.8
Hard	52.67	0.904	32.141	10.374	3.333	143274	5832768	1826020.3	490699.8
Expert	45.33	0.895	31.804	11.539	2.699	123038	4916839	1886465.3	307100.5

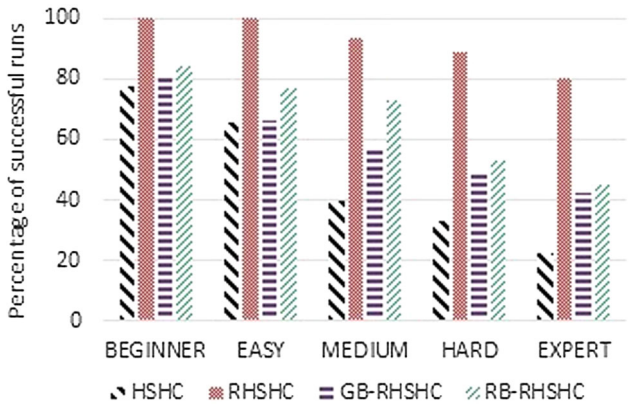

Fig. 3 Comparison in terms of success rate

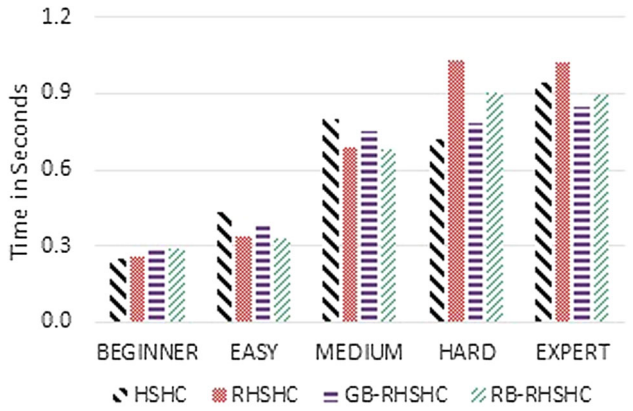

Fig. 4 Minimum execution time required

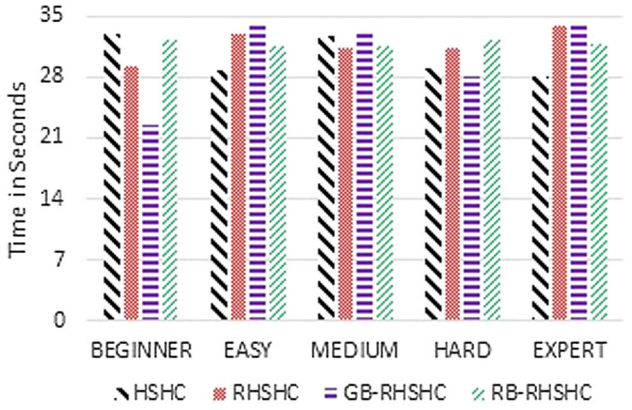

Fig. 5 Maximum execution time required

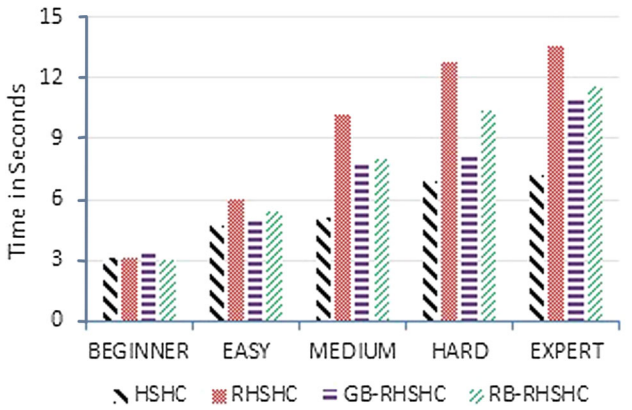

Fig. 6 Average execution time required

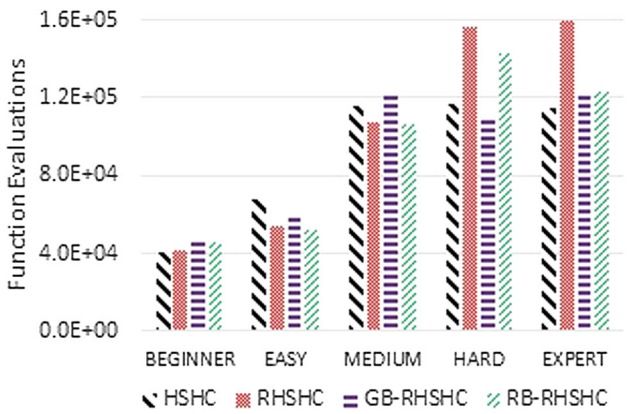

Fig. 7 Minimum number of function evaluations required

evaluations required to find the optimal. Note that success rate is the percentage of runs able to solve the Sudoku puzzle. It is evident from Fig. 3 that the order followed by four algorithms in terms of success rate is:

RHSHC > RB-RHSHC > GB-RHSHC > HSHC

Except for Beginner type puzzles the algorithms follow the same order in terms of execution time of successful runs. Now let us analyse the behavior of these algorithms. The reason for highest success rate and execution time of RHSHC is while trying to figure out global optima the algorithm extensively reinitialize its HM thus increasing the probability of escaping from local optima but at the same time increasing its execution time because the algorithm is not using its previous experience while further exploring the search space. While trying to escape from local optima by reinitializing HM the GB-RHSHC algorithm preserves the best harmony obtained so far and thus utilizes the best of its experience while as RB-RHSHC preserves one of the good harmonies and not necessarily the best thus former increases the probability of fast convergence than latter on the cost of increasing the probability of being stuck in local optima. HSHC never performs catastrophic mutation (i.e., does not unsettle its HM by reinitialization) thus increasing its convergence speed but at the same time decreasing the probability of escaping from local optima. Since beginner type Sudoku puzzles comparatively take less number of function evaluations as a result frequency of HM reinitialization is decreased, so all the four algorithms have almost same execution time for such type of problems.

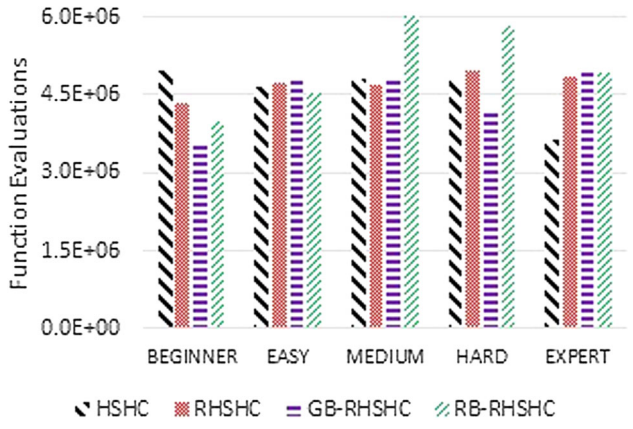

Fig. 8 Maximum number of function evaluations required

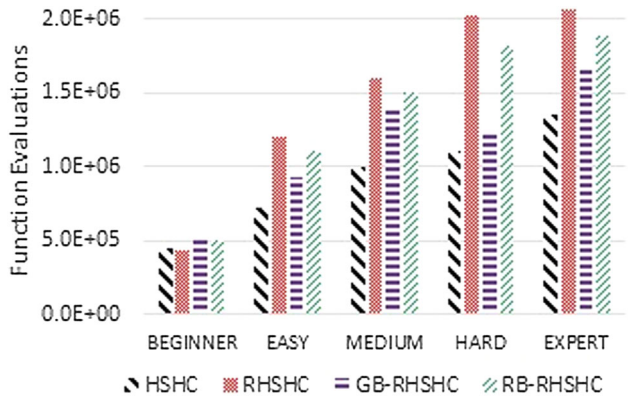

Fig. 9 Average number of function evaluations required

6 Comparison with other heuristic algorithms

The best performing RHSHC has been compared with the standard Harmony Search algorithm (Geem 2007), Hill Climber algorithm (Moraglio et al. 2006), Retrievable Genetic algorithm (Das et al. 2012) and hybrid AC3-tabu search algorithm (Soto et al. 2013) for solving sudoku. A brief introduction of the above mentioned algorithms has already been provided in Sect. 3. Retrievable Genetic algorithm has already been compared with Genetic algorithm proposed in Mantere and Koljonen (2006) and it has been established that former is superior to latter both in terms of effectiveness and efficiency so we have not compared our results with latter.

Since the fitness function proposed in this paper is different as used in Geem (2007) further the mechanism for generating new Harmony whether by memory consideration or by randomization is also different here, so in order to keep the comparison fair all the programs have been run for equal amount of time (35 seconds). It has been found that the HS algorithm to solve Sudoku proposed in Geem (2007) was not able to solve a single problem from the set of 25 Sudoku puzzles, thus all the four algorithms proposed in this paper (HSHC, RHSHC, GB-RHSHC, RB-RHSHC) significantly outperformed it.

The Hill climber algorithm proposed in Moraglio et al. (2006) is able to solve Easy sudoku puzzles with 100% success rate however completely failed in solving Medium and Hard puzzles. Thus RHSHC is very effective than standard Hill climber algorithm proposed in Moraglio et al. (2006) particularly for Medium, Hard and Expert level puzzles.

In order to keep the comparison fair between RHSHC and Retrievable GA (Das et al. 2012), Retrievable GA have been tested on same set of 25 sudoku puzzles, with the same stopping criteria and on the same machine on which RHSHC was executed. Further Retrievable GA algorithm is tested 30 time on each puzzle as was done with RHSHC algorithm.

Figures 10 and 11 compare the two algorithms (RHSHC and Retrievable GA) in terms of success rate and execution time respectively. As is evident from Fig. 10 the success rate of both algorithms is almost same for Beginner and Easy level puzzles however for Medium, Hard and Expert level puzzles RHSHC significantly out performs Retrievable GA. The difference is more evident in Expert level puzzles where the success rate of RHSHC is approximately 80% and that of Retrievable GA is only 7%. In terms of execution time Retrievable GA outperforms RHSHC for Beginner and Easy type puzzles, however for Medium, Hard and Expert level puzzles RHSHC outperforms Retrievable GA (Fig. 11). Thus RHSHC is the better performing algorithm (in terms of effectiveness as well as efficiency) than Retrievable GA particularly for Medium, Hard and Expert level puzzles.

Hybrid AC3-tabu search algorithm and RHSHC algorithm has been tested on the same set of 25 Sudoku puzzles, run on same machine and with same stopping criteria. Figures 12 and 13 compare the two algorithms (RHSHC and Hybrid AC3-tabu search) in terms of success rate and execution time respectively. As is evident from Figure 12

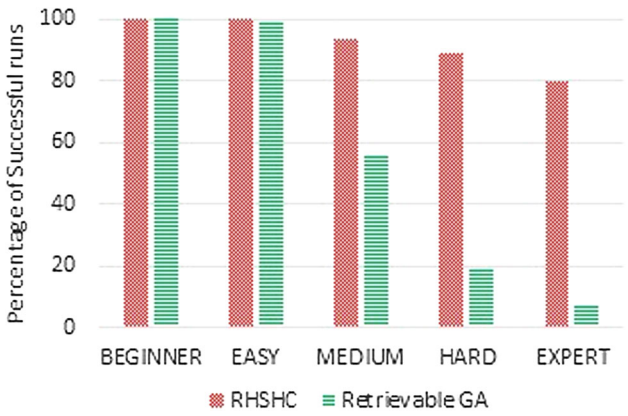

Fig. 10 Comparison of RHSHC and retrievable GA (success rate)

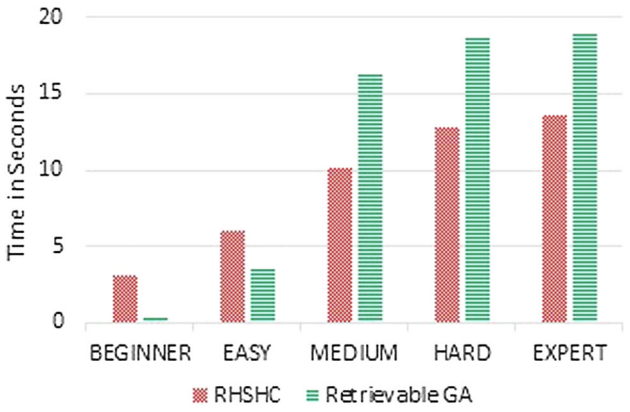

Fig. 11 Comparison of RHSHC and retrievable GA (execution time)

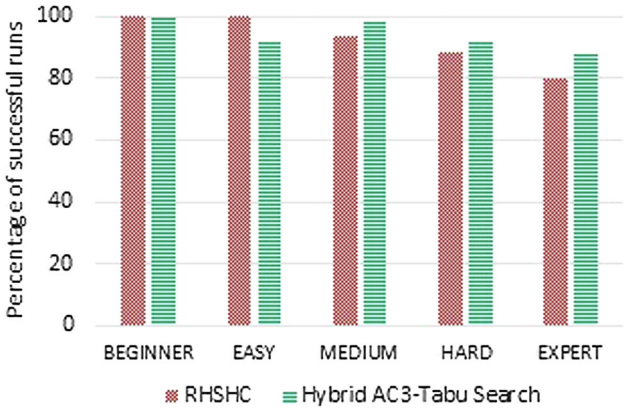

Fig. 12 Comparison of RHSHC and hybrid AC3-tabu search (success rate)

the success rate of both algorithms is almost same for Beginner level puzzles. RHSHC is slightly better than Hybrid AC3-tabu algorithm on Easy level puzzles whereas on Medium, Hard and Expert level puzzles hybrid AC3-tabu search algorithm has slight advantage over RHSHC. In terms of time complexity (Fig. 13) hybrid AC3-tabu search outperforms RHSHC on all category of puzzles, except for Easy type puzzles were both take approximately same time.

7 Wilcoxon's rank-sum test

A pair wise Wilcoxon's rank-sum test at 5% level of significance is used to statistically compare the performance of the competing algorithms. The sampling data used for applying the test has been obtained by performing 30 independent runs of each algorithm. The Wilcoxon's rank-sum test revealed that there is no statistically significant difference between RHSHC and Retrievable GA on Beginner and Easy type puzzles, however the superior performance of RHSHC over Retrievable GA is

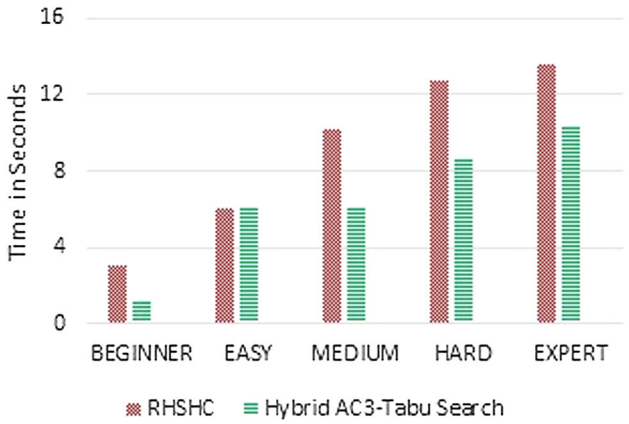

Fig. 13 Comparison of RHSHC and hybrid AC3-tabu search (execution time)

statistically significant on Medium, Hard and Expert level puzzles. Comparing the performance of RHSHC and Hybrid AC3-tabu search algorithm Wilcoxon's rank-sum test revealed that there is no statistically significant difference on Beginner, Medium and Hard type puzzles, however the better performance of RHSHC over AC3-tabu search algorithm is statistically significant on Easy type puzzles similarly the better performance of AC3-tabu search over RHSHC is statistically significant on Expert level puzzles.

8 Conclusion

This paper introduces a specialized Memetic algorithm namely HSHC created by hybridization of Harmony Search Algorithm and Hill Climbing operator to solve Sudoku puzzles. In order to increase exploration capabilities of HSHC algorithm three variations of basic HSHC are introduced. All the four proposed algorithms performed significantly better than the standard Harmony Search algorithm and standard Hill Climber algorithm. RHSHC outperformed its three variations (HSHC, GB-RHSHC, RB-RHSHC) in terms of success rate and was able to solve Beginner and Easy type problems with 100% success, further it also performed extremely well in solving Medium, Hard and Expert level puzzles.

Comparing RHSHC and Retrievable GA, it was established that former significantly outperformed latter in terms of effectiveness and efficiency particularly for Medium, Hard and Expert level puzzles. Experimental results demonstrate that RHSHC is competent (if not better) to recently proposed hybrid AC3-tabu search algorithm.

The objective of this paper is to study how a simple local search technique can be incorporated in a meta-heuristic algorithm to solve a complex problem like

Sudoku. In the future we intend to refine RHSHC algorithm so that its effectiveness particularly for Hard and Expert level puzzles is increased.

Acknowledgements The first author would like to acknowledge QIP Centre Indian Institute of Technology Roorkee, India and All India Council for Technical Education (AICTE) for sponsoring his research.

References

Takayuki Y, Takahiro S (2003) Complexity and completeness of finding another solution and its application to puzzles. IEICE Trans Fundam Electron Commun Comput Sci 86(5):1052–1060

Mantere T, Koljonen J (2006) Solving and rating sudoku puzzles with geneticalgorithms. In: New developments in artificial intelligence and the semantic web, proceedings of the 12th finnish artificial intelligence conference STeP. Citeseer, 2006, pp 86–92

Jones SK, Roach PA, Perkins S (2008) Construction of heuristics for a search-based approach to olving sudoku. In: Research and development in intelligent systems XXIV. Springer, 2008, pp. 37–49

Geem ZW, Kim JH, Loganathan G (2001) A new heuristic optimization algorithm: harmony search. Simulation 76(2):60–68

Gholizadeh S, Barzegar A (2013) Shape optimization of structures for frequency constraints by sequential harmony search algorithm. Eng Optim 45(6):627–646

Wang L, Li L-P (2013) An effective differential harmony search algorithm for the solving non-convex economic load dispatch problems. Int J Electr Power Energy Syst 44(1):832–843

Nekooei K, Farsangi MM, Nezamabadi-Pour H, Lee KY (2013) An improved multi-objective harmony search for optimal placement of dgs in distribution systems. Smart Grid IEEE Trans 4(1):557–567

Hadwan M, Ayob M, Sabar NR, Qu R (2013) A harmony search algorithm for nurse rostering problems. Inf Sci 233:126–140

Diao R, Shen Q (2012) Feature selection with harmony search. Syst Man Cybern Part B Cybern IEEE Trans 42(6):1509–1523

Fattahi H, Gholami A, Amiribakhtiar MS, Moradi S (2015) Estimation of asphaltene precipitation from titration data: a hybrid support vector regression with harmony search. Neural Comput Appl 26(4):789–798

Al-Betar MA, Khader AT, Zaman M (2012) University course timetabling using a hybrid harmony search metaheuristic algorithm. Syst Man Cybern Part C Appl Rev IEEE Trans 42(5):664–681

Geem ZW (2005) Harmony search in water pump switching problem. In: Proceedings of international conference on natural computation. Springer, pp. 751–760

Ong Y-S, Lim M-H, Zhu N, Wong K-W (2006) Classification of adaptive memetic algorithms: a comparative study. Syst Man Cybern Part B Cybern IEEE Trans 36(1):141–152

Dawkins R (2006) The selfish gene. Oxford university press, no. 199

Ong Y-S, Nguyen Q-H, Lim M-H, Jing T (2006) A development platform for memetic algorithm design. In: SCIS and ISIS 2006. Japan society for fuzzy theory and intelligent informatics, pp 1027–1032

Ishibuchi H, Yoshida T, Murata T (2003) Balance between genetic search and local search in memetic algorithms for multiobjective permutation flowshop scheduling. Evolut Comput IEEE Trans 7(2):204–223

Chan T-M, Leung K-S, Lee K-H (2012) Memetic algorithms for de novo motif discovery. Evolut Comput IEEE Trans 16(5):730–748

Sharma H, Bansal JC, Arya KV, Yang X-S (2016) Lévy flight artificial bee colony algorithm. Int J Syst Sci 47(11):2652–2670

Jadon SS, Bansal JC, Tiwari R, Sharma H (2015) Accelerating artificial bee colony algorithm with adaptive local search. Memet Comput 7(3):215–230

Sharma H, Bansal JC, Arya K (2013) Power law-based local search in differential evolution. Int J Comput Intell Stud 2(2):90–112

Hart WE, Krasnogor N, Smith JE (2005) Recent advances in memetic algorithms, vol 166. Springer Science & Business Media, New York

Moon TK, Gunther JH (2006) Multiple constraint satisfaction by belief ropagation: an example using sudoku. In: IEEE mountain workshop on 2006. Adaptive and learning systems, IEEE, 2006, pp. 122–126

Lynce I, Ouaknine J (2006) Sudoku as a sat problem. In: Proceedings of the 9th Symposium on Artificial Intelligence and Mathematics (AIMATH), 6 jan 2006

Lewis R (2007) Metaheuristics can solve sudoku puzzles. J Heuristics 13(4):387–401

Mullaney D (2006) Using ant systems to solve sudoku problems. University College Dublin, Dublin

Boryczka U, Juszczuk P (2012) Solving the sudoku with the differential evolution. Zeszyty Naukowe Politechniki Białostockiej. Informatyka, pp 5–16

Moon TK, Gunther JH, Kupin JJ (2009) Sinkhorn solves sudoku. Inf Theory IEEE Trans 55(4):1741–1746

Gunther J, Moon T (2012) Entropy minimization for solving sudoku. Signal Process IEEE Trans 60(1):508–513

Garey M, Johnson D (1979) Computers and intractability WH freeman and company New York

Das KN, Bhatia S, Puri S, Deep K (2012) A retrievable ga for solving sudoku puzzles. Tech. Rep, Citeseer

Nicolau M, Ryan C (2006) Solving sudoku with the gauge system. In: Genetic programming. Springer, pp 213–224

Li Y, Deng X (2011) Solving sudoku puzzles based on improved genetic algorithm. Jisuanji Yingyong Yu Ruanjian 28(3):68–70

Sato Y, Inoue H (2010) Solving sudoku with genetic operations that preserve building blocks. In: IEEE Symposium on Computational intelligence and games (CIG), 2010. IEEE, pp 23–29

Deng XQ, Da Li Y (2013) A novel hybrid genetic algorithm for solving sudoku puzzles. Optim Lett 7(2):241–257

Sato Y, Hasegawa N, Sato M (2013) Acceleration of genetic algorithms for sudoku solution on many-core rocessors. In: Massively parallel evolutionary computation on GPGPUs. Springer, pp 421–444

Moraglio A, Togelius J, Lucas S (2006) Product geometric crossover for the sudoku puzzle. In: IEEE congress on evolutionary computation, 2006. CEC 2006. IEEE, pp 470–476

Soto R, Crawford B, Galleguillos C, Paredes F, Norero E, (2015) A hybrid alldifferent-tabu search algorithm for solving sudoku puzzles. Comput Intell Neurosci 2015

Wang Z, Yasuda T, Ohkura K, (2015) An evolutionary approach to sudoku puzzles with filtered mutations. In: IEEE congress on evolutionary computation (CEC), 2015. IEEE, pp 1732–1737

Soto R, Crawford B, Galleguillos C, Monfroy E, Paredes F (2013) A hybrid ac3-tabu search algorithm for solving sudoku puzzles. Exp Syst Appl 40(15):5817–5821

Simonis H (2005) Sudoku as a constraint problem. In: CP workshop on modeling and reformulating constraint satisfaction problems. Citeseer, vol 12, pp 13–27

Rossi F, Van Beek P, Walsh T (2006) Handbook of constraint programming. Elsevier, Amsterdam

Manter T, Koljonen J (2007) Solving, rating and generating sudoku puzzles with ga. In: IEEE Congress on evolutionary computation, CEC 2007. IEEE 2007, pp 1382–1389

Soto R, Crawford B, Galleguillos C, Monfroy E, Paredes F (2014) A prefiltered cuckoo search algorithm with geometric operators for solving sudoku problems, vol 2014. The Scientific World Journal

Geem ZW (2007) Harmony search algorithm for solving sudoku. In: Knowledge-based intelligent information and engineering systems. Springer, pp 371–378

Weyland D (2015) A critical analysis of the harmony search algorithm how not to solve sudoku. Op Res Perspect 2:97–105

Durstenfeld R (1964) Algorithm 235: random permutation. Commun ACM 7(7):420

Jin X, Li Z (1997) Genetic-catastrophic algorithms and its application in nonlinear control system. J Syst Simul 9(2):111–115