Binary 100-Digit Challenge using IEEE-754 Coded Numerical Optimization Scenarios (100b-Digit) and V-shape Binary Distance-based Success History Differential Evolution (DISHv)

Aleš Zamuda
University of Maribor
Maribor, Slovenia
ales.zamuda@um.si

ABSTRACT

This paper proposes a new discrete optimization benchmark 100b-Digit, a binary discretized version for the 100-Digit Challenge. The continuous version 100-Digit Challenge utilizing continuous input parameters for a fitness function was suggested for the competitions at 2019 GECCO and 2019 CEC, while this paper proposes an extension, a discrete version of the 100-Digit Challenge, discretizing using an IEEE-754 double precision floating-point representation for the search space variables.

Furthermore, the paper also presents a viable recent state-of-the-art algorithm discretization to be applicable on the new 100b-Digit benchmark. V-shape transfer function is applied for binarization of the variables from the Distance-based Success History Differential Evolution (DISH) to create the new DISHv algorithm, which is then run on the 100b-Digit benchmark. The preliminary results for the DISHv algorithm are then reported for a basic value-to-reach termination criterion over 100b-Digit functions. This combination of 100b-Digit benchmark and DISHv algorithm demonstrates how to bridge a gap between recent continuous optimizers and different discrete benchmarks, and vice-versa.

CCS CONCEPTS

• **Mathematics of computing** → **Evolutionary algorithms**; *Bio-inspired optimization*; Nonparametric statistics; • **Theory of computation** → **Nonconvex optimization**; **Bio-inspired optimization**; *Stochastic control and optimization*; • **Computing methodologies** → *Continuous space search*; • **General and reference** → *Evaluation*; *Performance*;

KEYWORDS

continuous optimization, 100-digit challenge, large population size, Differential Evolution, DISH

GECCO '19 Companion, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-1-4503-6748-6/19/07…$15.00
https://doi.org/10.1145/3319619.3326898

ACM Reference Format:
Aleš Zamuda. 2019. Binary 100-Digit Challenge using IEEE-754 Coded Numerical Optimization Scenarios (100b-Digit) and V-shape Binary Distance-based Success History Differential Evolution (DISHv). In *Genetic and Evolutionary Computation Conference Companion (GECCO '19 Companion), July 13–17, 2019, Prague, Czech Republic.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3319619.3326898

1 INTRODUCTION

This paper proposes a new discrete optimization benchmark, 100b-Digit, a binary discretized version for the 100-Digit Challenge. The continuous version 100-Digit Challenge [23], which utilizes continuous input parameters for a fitness function, was suggested for the most recent 2019 GECCO and 2019 CEC competitions, while this paper proposes an extension, a discrete version of the 100-Digit Challenge. This paper, hence, proposes discretizing the numbers using an IEEE-754 double precision floating-point representation [17] for the search space of continuous 100-Digit Challenge functions. By doing this it brings closer continuous and discrete optimization algorithm development and also proposes an update to recent and challenging benchmark following the budget-free largely solved 100-Digit Challenge [31]. Therefore, as because it is dealing with discrete optimization, this paper is associated with the 2019 GECCO BB-DOB Workshop, which seeks to cope with the discrete domain and its application scenarios. The new 100b-Digit benchmark is one such application scenario, set under research in benchmarking. This benchmark is target-based and budget-free, similar to BBOB with COCO [12].

The statistics of 100b-Digit are the same as for the 100-Digit Challenge [23], i.e., ten test functions' sums of optimum fitness matched digits up to ten (hence the "100" in the name), in the best half performed trial runs of an algorithm. Both 100-Digit challenges are value-to-reach based. This performance criterion has an advantage of being independent from unrelated alternatives, as opposed to several ranking-based measures used in some other benchmarks.

Furthermore, the paper also presents demonstrative optimizer algorithm for the new benchmark, a viable recent state-of-the-art algorithm discretization to be applicable on the new 100b-Digit benchmark. The V-shape transfer function V4 from [15] is applied for binarization of the variables from the Distance-based Success History Differential Evolution (DISH) [30] to create the new DISHv algorithm which is then run on 100b-Digit benchmark. The DISHv algorithm results are then reported for a basic maximum number of fitness termination criterion over 100b-Digit functions. The DISHv algorithm is based on the DISHchain algorithm [31], which was assessed on the original benchmark [23].

Related work is presented in the following Section. Section 3 presents the proposed benchmark and new DISHv algorithm. Section 4 reports results as defined in the proposed benchmark. Section 5 provides conclusions with suggestions for future work.

2 RELATED WORK

In this section, the 100-Digit Challenge is explained in the next subsection, and then the works from the applied population-based optimization algorithm are covered in the second subsection.

2.1 100-Digit Challenge

As motivated within the technical report [23], "research on single objective optimization algorithms often forms the foundation for more complex scenarios, such as niching algorithms and both multi-objective and constrained optimization algorithms." The usefulness of such research is demonstrated clearly for approaches in complex mission planning, like underwater glider path planning in deep ocean scenarios [38, 39], or energy scheduling [10] and spatial computer vision [34].

As indicated further for the challenge [23], "Traditionally, single objective benchmark problems are also the first test for new evolutionary and swarm algorithms.", and this can be demonstrated over a plethora of algorithms, including the most recent widely applied SHADE algorithm [26] variants [2, 5, 11, 28, 30, 38].

The goal of benchmarking on the 100-Digit challenge and also this paper is to understand better "the behavior of swarm and evolutionary algorithms as single objective optimizers" [23] and, hence, this paper introduces a modified DISH algorithm to a binary 100-Digit Challenge.

Further motivated in [23], "The SIAM 100-Digit Challenge was developed in 2002 by Nick Trefethen in conjunction with the Society for Industrial and Applied Mathematics (SIAM) as a test for high-accuracy computing" [3, 29], specifically, "the challenge was to solve 10 hard problems to 10 digits of accuracy" and, in a similar vein, the 100-Digit Challenge is proposed in [23].

The name of the SIAM 100-Digit Challenge is due to as [23] puts it: "One point was awarded for each correct digit, making the maximum score 100". The new "100-Digit Challenge asks contestants to solve all ten problems with one algorithm, although limited control parameter tuning for each function is permitted to restore some of the original contest's flexibility", and it defines that the "difference is that the score for a given function is the average number of correct digits in the best 25 out of 50 trials.".

The discretization, as will be presented in next section, is motivated by [15], where a 15-bit integer without floating point is binarized on a 5-dimension continuous function, reaching bit resolutions maximally up to 10x15=150 bits (mantissa only), and in this paper a double-precision floating-point representation is added (adding the exponent), shown on the 100-Digit Challenge functions.

2.2 Population-based Optimization

The base algorithm of the DISHv algorithm is the DISH algorithm [30], which already includes the computational mechanisms like those from the basic architecture of Differential Evolution (DE) [25] (population-based optimization, mutation, crossover, and selection over floating-point vectors), the Success-History based Adaptive Differential Evolution (SHADE) algorithm [26] (parameter adaptation with external archive and historical control parameter memory), and its updates L-SHADE [28] (linear decrease in population size) and jSO [5] (parameterization and weighted mutation).

More about the population based optimization and DE development can be read in reviews like [1, 7, 18, 21, 22, 27]. As covered recently in [38], similarly for L-SHADE, DISH is also an extension of the basic Differential Evolution (DE), as a floating-point encoding evolutionary algorithm [9] for global optimization over continuous spaces.

Several recent surveys and insights exist with the DE algorithm's base name [6–8, 16, 21, 35] and its metaphors [4, 24], stemming from the progress on computational mechanisms, mainly from the branches of the DE, as well as applications [7, 10, 20, 34, 38]. Binary versions of DE have been published before in the works like [13, 14, 19].

The basic DE [25] consists of an evolutionary loop, within which are evolved new population D-dimensional population vectors $\mathbf{x}_i$, $\forall i \in \{1, 2, ..., NP\}$. During each generation step number $g \in \{1, 2, ..., G\}$, computational operators like mutation, crossover, and selection of the population are performed, until a termination criterion is satisfied, like a fixed number of maximum fitness evaluations (*MAX_FES*). L-SHADE and the DISH extend DE with population size reduction [28], that was already introduced to DE in continuous optimization and real-world industry challenges [33], and a well performing parameter control mechanism, named Success History (SH) [28]. The DISH algorithm's underlying L-SHADE variants have won several recent evolutionary benchmarking competitions [2, 5, 11, 26, 28].

Given that the benchmark [23] contains the code for L-SHADE in C++ language, and that [31] is also written in C++ language (as well as [5]), in the following the listings are provided for code changes from DISHchain to DISHv. Namely, the core algorithm code file is unchanged (`lshade.cc`), only file `search_algorithm.cc` (seen in Fig. 4 is updated for the specific search algorithm.

3 PROPOSED 100B-DIGIT BENCHMARK AND DISHV ALGORITHM

This section introduces the new discrete benchmark 100b-Digit in the next subsection, and then the subsection following it demonstrates a new algorithm, DISHv, used with this benchmark.

3.1 New Benchmark: 100b-Digit

The new benchmark, 100b-Digit, presents discretizing 100-Digit Challenge with IEEE-754 binary coded numerical optimization scenarios. In Table 1 the new 100b-Digit challenge basic test functions based on the 100-Digit Challenge functions and features are provided, addressing deep understanding of the problem at hand and the features that make instances of the problem hard for these algorithms. The properties of the functions and binarization parameters are, hence, listed in Table 1.

The first column in Table 1 lists the consecutive problem number (No.), which is same as for the 100-Digit Challenge [23]. The second column in Table 1 highlights the number of required trials for each problem; it is a constant at 50 for each problem. The third column in Table 1 is the name of the problem on which a test function is

Figure 1: DISHv algorithm pseudocode with evaluation and binarization perspectives: V4 transfer function binarized DIstance-Based Success History Differential Evolution. The pseudocode outline is based on original DISH pseudocode (for continuous optimization), see [30].

Set NP_{init}, NP_f, H, and stopping criterion;
$NP = NP_{\text{init}}$, $G = 0$, $\boldsymbol{x}_{\text{best}} = \{\}$, $k = 1$, $p_{\min} = 2/NP$, $\boldsymbol{A} = \emptyset$;
Randomly initialize population $\boldsymbol{P} = (\boldsymbol{x}_{1,G},\ldots,\boldsymbol{x}_{NP,G})$ where $\forall i : \mathbf{x}_{i,G} = \{x_{1,i}, x_{2,i}, ..., x_{D,i}\}$ in generation G;
Binarize: $\forall_{i,j} : x^{\text{B}}_{i,j} = \mathcal{U}[0,1] < |(\frac{2}{\pi} \arctan(\frac{\pi}{2} x_{i,j})|$
Evaluate $\forall_i f(\mathbf{x}_{i,G}) := f^{\text{B}}(\mathbf{x}^{\text{B}}_{i,G})$
Set all values in $\mathbf{M}_{\text{F}}$ to 0.5 and $\mathbf{M}_{\text{CR}}$ to 0.8;
$\boldsymbol{P}_{\text{new}} = \{\}$, $\boldsymbol{x}_{\text{best}}$ = best from population $\boldsymbol{P}$;
while stopping criterion not met **do**
  $\mathbf{S}_F = \emptyset$, $\mathbf{S}_{CR} = \emptyset$;
  for $i = 1$ to NP **do**
    $r = \mathcal{U}[1, H]$;
    if $r = H$ **then**
      $M_{\text{F},r} = M_{\text{CR},r} = 0.9$;
    end if
    if $M_{\text{CR},r} < 0$ **then**
      $CR_{i,G} = 0$;
    else
      $CR_{i,G} = \mathcal{N}(M_{\text{CR},r}, 0.1)$;
    end if
    $F_{i,G} = C\left[M_{\text{F},r}, 0.1\right]$;
    if $G < 0.6G_{\text{MAX}}$ **and** $F_{i,G} > 0.7$ **then**
      $F_{i,G} = 0.7$;
    end if
    if $G < 0.25G_{\text{MAX}}$ **then**
      $CR_{i,G} = \max(CR_{i,G}, 0.7)$;
    else if $G < 0.5G_{\text{MAX}}$ **then**
      $CR_{i,G} = \max(CR_{i,G}, 0.6)$;
    end if
    $\boldsymbol{x}_{i,G} = \boldsymbol{P}[i]$, $p_i = U[p_{\min}, 0.2]$;
    $F_w = \begin{cases} 0.7 \times F, & FES < 0.2MAXFES, \\ 0.8 \times F, & FES < 0.4MAXFES, \\ 1.2 \times F, & \text{otherwise.} \end{cases}$
    $\mathbf{v}_{i,G} = \mathbf{x}_i + F_w(x_{p\text{Best}} - \mathbf{x}_i) + F(\mathbf{x}_{\text{r1}} - \mathbf{x}_{\text{r2}})$;
    $u_{j,i,G} = \begin{cases} v_{j,i} & \text{if } \mathcal{U}[0,1] \le CR_i \text{ or } j = j_{\text{rand}} \\ x_{j,i} & \text{otherwise} \end{cases}$;
    Binarize: $\forall_{i,j} : u^{\text{B}}_{i,j} = \mathcal{U}[0,1] < |(\frac{2}{\pi} \arctan(\frac{\pi}{2} u_{i,j})|$
    Evaluate $\forall_i f(\mathbf{u}_{i,G}) := f^{\text{B}}(\mathbf{u}^{\text{B}}_{i,G})$
    if $f^{\text{B}}(\boldsymbol{u}_{i,G}) \le f^{\text{B}}(\boldsymbol{x}_{i,G})$ **then**
      $\boldsymbol{x}_{i,G+1} = \boldsymbol{u}_{i,G}$;
    else
      $\boldsymbol{x}_{i,G+1} = \boldsymbol{x}_{i,G}$;
    end if
    if $f^{\text{B}}(\boldsymbol{u}_{i,G}) < f^{\text{B}}(\boldsymbol{x}_{i,G})$ **then**
      $\boldsymbol{x}_{i,G} \rightarrow \boldsymbol{A}$;
      $F_i \rightarrow \mathbf{S}_F$, $CR_i \rightarrow \mathbf{S}_{CR}$;
    end if
    if $|\boldsymbol{A}| > NP$ **then**
      Randomly delete $|\boldsymbol{A}| - NP$ individuals from $\boldsymbol{A}$;
    end if
    $\boldsymbol{x}_{i,G+1} \rightarrow \boldsymbol{P}_{\text{new}}$;
  end for
  Calculate NP_{new} according to:
  $NP_{\text{new}} = \text{round}\left(NP_0 - \frac{FES}{MAXFES}(NP_0 - NP_{\text{f}})\right)$;
  if $NP_{\text{new}} < NP$ **then**
    Sort individuals in $\boldsymbol{P}$ according to their objective function values and remove $NP - NP_{\text{new}}$ worst ones;
    $NP = NP_{\text{new}}$;
  end if
  if $|\boldsymbol{A}| > NP$ **then**
    Randomly delete $|\boldsymbol{A}| - NP$ individuals from $\boldsymbol{A}$;
  end if
  Update $M_{\text{F},k}$, $M_{\text{CR},k}$ using Lehmer mean:
  if $\mathbf{S}_F \neq \emptyset$ **and** $\mathbf{S}_{CR} \neq \emptyset$ **then**
    $w_k = \frac{\sqrt{\sum_{j=1}^{D}(u_{k,j,G} - x_{k,j,G})^2}}{\sum_{m=1}^{|S_{CR}|}\sqrt{\sum_{j=1}^{D}(u_{m,j,G} - x_{m,j,G})^2}}$
    $\text{mean}_{WL}(\text{S}) = \frac{\sum_{k=1}^{|\text{S}|} w_k \bullet S_k^2}{\sum_{k=1}^{|\text{S}|} w_k \bullet S_k}$;
    $M_{F,k} = \begin{cases} \text{mean}_{WL}(\text{S}_F) & \text{if } \text{S}_F \neq \emptyset \\ M_{F,k} & \text{otherwise} \end{cases}$
    $M_{CR,k} = \begin{cases} \text{mean}_{WL}(\text{S}_{CR}) & \text{if } \text{S}_{CR} \neq \emptyset \\ M_{CR,k} & \text{otherwise} \end{cases}$
    Increase $k := k + 1$ and if new $k > H$, reset k to 1;
  end if
  $\boldsymbol{P} = \boldsymbol{P}_{\text{new}}$, $\boldsymbol{P}_{\text{new}} = \{\}$, $\boldsymbol{x}_{\text{best}}$ = best from population $\boldsymbol{P}$, G++;
end while
return $\boldsymbol{x}_{\text{best}}$ as the best found solution;

based. The fourth column in Table 1 lists the number of continuous variables D_c the function exhibits with continuous search space of the original benchmark [23] before transformation to this new binary benchmark (100b-Digit). The fifth column in Table 1 defines the continuous search variable $\mathbf{x}$ limits. The sixth column in Table 1 defines the optimum of the functions, $F(\mathbf{x}^*)$, which is equal to 1 for each function in the benchmark, as in [23]. Columns 7-10 in Table 1 list features of the functions, like separable, multimodal, shifted, and rotated, respectively, demonstrating that the benchmark is comprised of complex optimization functions.

Columns 11-14 in Table 1 list the encoding aspect of the binarization transformation. Namely, the IEEE-754 compilant double precision floating-point numbers are comprised upon composition of these bits. The IEEE-754 encoding uses an exponent mask, which masks the bits of an otherwise 11-bit exponent down to 4-bit (functions 1 and 2), 3-bit (function 4–10), or 2-bit (function 3). This is because the exponent is defined by the power of 2, and, e.g., 4 bits

Figure 2: Discretization function under SHADE framework for the 100b-Digit Challenge.

```
// the part of a black-box function: use the CEC2019 to optimize over binary sequence
Fitness discretized_cec19_test_func(bool x[], double *f, int nx, int mx, int func_num) {
  static double continuousX[1000]; // increase the buffer size when D > 1000
  unsigned long long exponentLimitMask;
  // 12 bit limit => 8192 = 2 ^ 12 => exponent limit mask is 4-bit, since 2^4=16 >= 12
  if (func_num == 1) exponentLimitMask = 0b00000001111;
  else if (func_num == 2) exponentLimitMask = 0b00000001111;
  if (func_num == 3) exponentLimitMask = 0b0000000011;
  else exponentLimitMask = 0b00000000111;

  for (int i = 0; i < mx; i++) {
    for (int j = 0; j < nx/64; j++)
      continuousX[j] = bitsToDoubleIEEE754(x + 64*(j + nx*i), exponentLimitMask);

    cec19_test_func(continuousX, &f[i], nx/64, 1, func_num);
  }
}
```

Figure 3: Utility functions for IEEE-754 double-precision calculations.

```
unsigned long bitArrayToInt64(bool bits[], int count) {
  unsigned long converted = 0;
  unsigned long tmp;
  for (int i = 0; i < count; i++) {
    tmp = bits[i];
    converted |= tmp << (count - i - 1); // collect the bits
  }
  return converted;
}

double bitsToDoubleIEEE754(bool x[], const unsigned long long &exponentLimitMask) {
  unsigned long long sign = x[0]; // if 1 the number is negative
  unsigned long long exponent = bitArrayToInt64(x+1, 11);
  unsigned long long mantissa = bitArrayToInt64(x+12, 52);

  exponent &= exponentLimitMask; // apply mask
  exponent += 0b01111111111; // set bit 62 to 1 (+1023)

  unsigned long long bits64num = (sign << 63) | (exponent << 52) | mantissa; // compose binary64

  return *reinterpret_cast<double*>(&bits64num);
}
```

Figure 4: DISHv algorithm code patch under SHADE framework.

```
void searchAlgorithm::evaluatePopulation(const vector<Individual> &pop, vector<Fitness> &fitness) {
  for (int i = 0; i < pop_size; i++) {
    // convert to binary before evaluation - on the part of the EA
    static bool transferredXbinarized[64*1000]; // increase buffer when D > 1000
    for (int j = 0; j < problem_size; j++) {
      double transferredX = fabs(2/PI * atan(PI/2*pop[i][j])); // apply V-shape
      bool binT = randDouble() < transferredX; // binarize
      transferredXbinarized[j] = binT;
    }
    discretized_cec19_test_func(transferredXbinarized, &fitness[i], problem_size, 1, function_number);
  }
}

void searchAlgorithm::initializeFitnessFunctionParameters() {
  // Discretizes CEC2019 transfer function
  max_region = 8;
  min_region = -8;
  //epsilon is an acceptable error value
  epsilon = pow(10.0, -11);
  optimum = 1;
}
```

are sufficient to represent 2^4=16-bit exponent — i.e. numbers upto 16384 are representable using 4 bits, like for the range used for $\mathbf{x}$ using function 2.

Column 13 in Table 1 depicts how to calculate the bit-width of a search vector $|\mathbf{x}^{\mathrm{B}}|$. The last column in Table 1 then lists the effective dimension D of the new binarized functions. The binary functions' dimensions range from $D = 570$ upto $D = 1008$, which exceeds, e.g., the 15-bit integer without floating point when the binarized 5-dimension continuous function reaches bit resolutions maximally up to 10x15=150 bits in [15]. To calculate D, the number of continuous variables is multiplied by the bitstring length of a binary variable for each function from columns 11-13: 53 bits for mantissa, bits of the exponent, and a sign bit. Table 1 therefore provides further insight for developers of new algorithms for this benchmark. As seen from the columns for exponent and mantissa representtion, the binarization and masking used allow the problem variable to be encoded with high precision at low bit length, hence, making the information about the fitness function dense. By changing number of underlying continuous variables D_c and their definition intervals it would also be possible to change the dimensions and hence hardness of the problems, making the benchmark simply scalable.

To convert the C++ code of the benchmark to a binary version, the `search_algorithm.c` code of L-SHADE framework is updated with a patch, as listed in Fig. 2. The proposed benchmark uses a utility code, as listed in Fig. 3, to convert the bitstring to an IEEE-754 double, where a bit-mask is applied to cap the exponent (`exponent &= exponentLimitMask;`) in the `bitsToDoubleIEEE754` function. Namely, for each test function, the mantissa is the most important limiter of value range to search for and, hence, the 100b-Digit defines a bit-aligned range for the exponent encoding the double-precision floating-point. The `bits64num` denotes a list of 64 bits that are represented in a 64-bit `unsigned long long` number composed by shifting 64 bits b_k in place of the IEEE-754 double-precision floating-point format [17] bits representation on a little-endian machine. The `bits64num` is converted to C++ `double` type by calling reinterpret cast as `*reinterpret_cast<double*>(&bits64num)` for values of bits b_k, $\forall k = 0, 1, ..., 63$, taken from a D-dimenzional array of binarized DE vector values $\mathbf{x}^{\mathrm{B}} = \{b_1, b_2, ..., b_{63}\}^{D_c}$ that compositely represent D_c continuous variables $x_j, j = 1, 2, ..., D_c$ to be used when calling the original 100-Digit functions in Table 1:

$$\mathbf{x}_j \longleftarrow (-1)^{b_{63}} \left(1 + \sum_{k=1}^{52} 2^{-k} b_{52-k}\right) \times 2^{-1023+\sum_{k=52}^{62} 2^{k-52} b_k}, \quad (1)$$

while exponent bits ($k = \{52, 53, ..., 62\}$) are never all equal, therefore the IEEE-754 subnormal representation and NaN representation are both avoided. This is due to previously described masking with `exponentLimitMask` that keeps some zero bits at beginning (e.g. if a mask is $1111_{(2)}$, after masking 5 bits b_{57}, b_{58}, ..., b_{61} are 0) and $b_{62} = 1$.

Using function `discretized_cec19_test_func` in Fig. 2, the collection of discretized binary numbers takes place, putting the numbers together and calling a regular 100-Digit Challenge function. Note also, in `main.c`, the `g_problem_size_arr` elements are multiplied by 64, since each bit now has its own DE component.

Provided that there are numerical optimization competitions at CEC with a series of long-term track of DE optimization algorithm variants taking place and winning in most cases [32], a DE algorithm is also applied demonstratively to the new 100b-Digit benchmark for baseline, as described in next section. Also, due to the V4 shape [15], the search parameters for DISHv are limited to [-8,8] in `initializeFitnessFunctionParameters`, see Fig. 4.

3.2 DISHv Algorithm

The DISHv algorithm upgrades the DISHchain algorithm with discretization, as proposed with the V4 transfer function in V-shaped binarization functions from [15]. In order to define DISHv thoroughly, the pseudocode of the algorithm is provided in Fig.1 for the sake of clarity. The pseudocode outline is based on the original continuous DISH pseudocode [30]. Within the DISHv algorithm pseudocode, the perspectives of evaluation and binarization are additionally highlighted, therefore they are put in underline text font and blue color. The binarization with **V**4 transfer function is provided in lines 4 and 33, and the evaluation function use in lines 5 and 34. $\mathcal{U}[0, 1]$ denotes a uniform random continuous function between 0 and 1. Negative values are also a valid input to the arctan function [15]. The rest of the pseudocode in Fig.1 defines the DISH algorithm [30]: Lines 1–3 and 6–8 apply initial parameter settings and population initialization. Then the generations' loop at line 8 runs, selecting the DE mutation indexes in line 11, then applying control parameter adaptation in lines 12–30, and followed by mutation in line 31 and crossover in line 32. The new binarization and evaluation are again used within the generations' loop through lines 33–48, where selection operator is applied. Population structuring with archive strategy and population sizing then follows in lines 49–56. Then, lines 57–63 apply the distance-based control parameters' adaptation (hence the name of DISH). Line 64 switches generations' pointers, and line 66 then returns the best found solution.

Explained in the plain level of L-SHADE C++ codebase provided within demonstration files of the 100-Digit Challenge, in the following, the newly added (as of this paper) code functions, and other necessary changes, are listed as follows. To evaluate the population, L-SHADE code calls `evaluatePopulation` function, which binarizes the parameters effectively by transfer function (`transferredX = fabs(2/PI * atan(PI/2*pop[i][j]))`), and then calls the C++ function `discretized_cec19_test_func` to perform optimization over binary variables. The code listing is provided in Fig. 4.

4 RESULTS

The DISHv algorithm is assessed using the maximum number of fitness evaluations (*MAX_FES*) termination criterion. The fifty runs for each function sorted by the number of correct digits [23] is provided as Table 2 using *MAX_FES* = 1e+6, Table 3 using *MAX_FES* = 1e+7, Table 4 using *MAX_FES* = 1e+8, and Table 5 using *MAX_FES* = 1e+9.

As required in [23], over *MAX_FES*=1e+6, Table 2 (and Table 3 using *MAX_FES* =1e+7) "lists for each function the number of trials in a run of 50 that found n correct digits, where $n = 1, 2, ..., 10$. In the final column is entered the average number of correct digits in the best 25 runs, i.e. the score for that function. The total score is

Table 1: The new 100b-Digit Challenge Basic Test Functions, based on the 100-Digit Challenge, and their features [23].

No.	Trials	Name	D_c	**x** range	$F(\mathbf{x}^*)$	Separable	Multimodal	Shifted	Rotated	Exponent mask	Mantissa	$\lvert\mathbf{x}^B\rvert$	D
1	50	Storn's Chebyshev Polynomial Fitting Problem	9	[-8192, 8192]	1	N	Y	N	N	4-bit: $1111_{(2)}$	53-bit	9 x 58-bit	522
2	50	Inverse Hilbert Matrix Problem	16	[-16384, 16384]	1	N	Y	N	N	4-bit: $1111_{(2)}$	53-bit	16 x 58-bit	928
3	50	Lennard-Jones Minimum Energy Cluster	18	[-4,4]	1	N	Y	N	N	2-bit: $11_{(2)}$	53-bit	18 x 56-bit	1008
4	50	Rastrigin's Function	10	[-100,100]	1	N	Y	Y	Y	3-bit: $111_{(2)}$	53-bit	10 x 57-bit	570
5	50	Griewangk's Function	10	[-100,100]	1	N	Y	Y	Y	3-bit: $111_{(2)}$	53-bit	10 x 57-bit	570
6	50	Weierstrass Function	10	[-100,100]	1	N	Y	Y	Y	3-bit: $111_{(2)}$	53-bit	10 x 57-bit	570
7	50	Modified Schwefel's Function	10	[-100,100]	1	N	Y	Y	Y	3-bit: $111_{(2)}$	53-bit	10 x 57-bit	570
8	50	Expanded Schaffer's F6 Function	10	[-100,100]	1	N	Y	Y	Y	3-bit: $111_{(2)}$	53-bit	10 x 57-bit	570
9	50	Happy Cat Function	10	[-100,100]	1	N	Y	Y	Y	3-bit: $111_{(2)}$	53-bit	10 x 57-bit	570
10	50	Ackley Function	10	[-100,100]	1	N	Y	Y	Y	3-bit: $111_{(2)}$	53-bit	10 x 57-bit	570

Table 2: Fifty runs for each function sorted by the number of correct digits (for the DISHv algorithm), *MAX_FES*=1e+6.

Function	**Number of correct digits**											Score
	0	1	2	3	4	5	6	7	8	9	10	
1	50	0	0	0	0	0	0	0	0	0	0	0
2	50	0	0	0	0	0	0	0	0	0	0	0
3	50	0	0	0	0	0	0	0	0	0	0	0
4	50	0	0	0	0	0	0	0	0	0	0	0
5	50	0	0	0	0	0	0	0	0	0	0	0
6	50	0	0	0	0	0	0	0	0	0	0	0
7	50	0	0	0	0	0	0	0	0	0	0	0
8	50	0	0	0	0	0	0	0	0	0	0	0
9	50	4	0	0	0	0	0	0	0	0	0	0.16
10	50	0	0	0	0	0	0	0	0	0	0	0
											Total:	0.16

Table 3: Fifty runs for each function sorted by the number of correct digits (for the DISHv algorithm), *MAX_FES*=1e+7.

Function	**Number of correct digits**											Score
	0	1	2	3	4	5	6	7	8	9	10	
1	50	1	0	0	0	0	0	0	0	0	0	0.04
2	50	0	0	0	0	0	0	0	0	0	0	0
3	50	0	0	0	0	0	0	0	0	0	0	0
4	50	0	0	0	0	0	0	0	0	0	0	0
5	50	0	0	0	0	0	0	0	0	0	0	0
6	50	0	0	0	0	0	0	0	0	0	0	0
7	50	0	0	0	0	0	0	0	0	0	0	0
8	50	0	0	0	0	0	0	0	0	0	0	0
9	50	50	0	0	0	0	0	0	0	0	0	1
10	50	0	0	0	0	0	0	0	0	0	0	0
											Total:	1.04

entered (the sum of the scores for all 10 functions) in the bottom right-hand cell." The reported DISHv algorithm score over the experiments from Tables 2–5 is, hence, 0.04 for 1e+6, 1 for 1e+7, 2.08 for 1e+8, and 3.04 for 1e+9 *MAX_FES*, respectively. As seen with gradual increase of score, this new 100b-Digit benchmark therefore provides a much bigger challenge than the original CEC-75 100-Digit benchmark.

The reported results are preliminary and first benchmark results for an algorithm with this new benchmark. It is seen, that the benchmark is not easy to solve and shall pose interesting challenge ahead for researchers to tackle it. With possible further extended time [36], it is expected that adding higher values of *MAX_FES* would increase the DISHv algorithm score on the 100b-Digit benchmark. Therefore the obtained best fitness values $f(\mathbf{x}^B)$ at the end of each trial run of the DISHv algorithm using 1e+7, averaged over

Table 4: Fifty runs for each listed function sorted by the number of correct digits (for the DISHv algorithm), *MAX_FES*=1e+8.

Function	Number of correct digits											Score
	0	1	2	3	4	5	6	7	8	9	10	
1	50	49	2	0	0	0	0	0	0	0	0	1.08
2	50	0	0	0	0	0	0	0	0	0	0	0
3	50	0	0	0	0	0	0	0	0	0	0	0
4	50	0	0	0	0	0	0	0	0	0	0	0
5	50	0	0	0	0	0	0	0	0	0	0	0
6	50	0	0	0	0	0	0	0	0	0	0	0
7	50	0	0	0	0	0	0	0	0	0	0	0
8	50	0	0	0	0	0	0	0	0	0	0	0
9	50	50	0	0	0	0	0	0	0	0	0	1
10	50	0	0	0	0	0	0	0	0	0	0	0
											Total:	2.08

Table 5: Fifty runs for each listed function sorted by the number of correct digits (for the DISHv algorithm), *MAX_FES*=1e+9.

Function	Number of correct digits											Score
	0	1	2	3	4	5	6	7	8	9	10	
1	50	50	50	0	0	0	0	0	0	0	0	2
2	50	0	0	0	0	0	0	0	0	0	0	0
3	50	0	0	0	0	0	0	0	0	0	0	0
4	50	0	0	0	0	0	0	0	0	0	0	0
5	50	0	0	0	0	0	0	0	0	0	0	0
6	50	0	0	0	0	0	0	0	0	0	0	0
7	50	0	0	0	0	0	0	0	0	0	0	0
8	50	0	0	0	0	0	0	0	0	0	0	0
9	50	50	1	0	0	0	0	0	0	0	0	1.04
10	50	0	0	0	0	0	0	0	0	0	0	0
											Total:	3.04

Table 6: Obtailed best fitness values $f(\mathbf{x}^{\mathbf{B}})$ at the end of each trial run of the DISHv algorithm using 1e+7, averaged over 50 runs.

Function	Average $f^{\mathrm{B}}(\mathbf{x}^{\mathrm{B}}) - 1$	Standard deviation of $f^{\mathrm{B}}(\mathbf{x}^{\mathrm{B}})$
1	5.177e+00	4.527e+00
2	9.134e+00	1.488e+00
3	8.514e+00	4.059e-01
4	5.634e+01	6.408e+00
5	1.865e+01	5.228e+00
6	7.080e+00	4.757e-01
7	1.241e+03	1.274e+02
8	3.569e+00	1.713e-01
9	6.451e-01	1.054e-01
10	2.015e+01	3.426e-02

50 runs, are listed demonstratively in Table 6 for reference values. Moreover, as the obtained scores are still very low, this suggests that more interesting benchmarking can be applied using the likes of the proposed DISHv and 100b-Chain combination.

5 CONCLUSIONS

This paper proposed a new 100-Digit Challenge, the binarized 100b-Digit Challenge. Then, the algorithm DISHv was proposed and assessed on this new 100b-Digit Challenge on Single Objective Numerical Optimization. The DISHv algorithm results were presented according to the specification of report [23]. The new benchmark consists of problems of very high dimension (522 – 1008), which makes them very challenging after discretization from underlying continuous space. This is clearly visible in Tables 2-5, where hardly any problems were solved with accuracy of more than 1 digit, despite using a variant of the state-of-the-art DE algorithm. With the original 100-Digit Challenge possibly being solved soon [31], the new 100b-Digit Challenge still remains largely unsolved.

By describing how to discretize a real test benchmark using limited exponent mask for floating-point variables through IEEE-754 binarization, and then how to apply a transfer function over an existing continuous optimizer to enhance it to become applicable on discrete problems, this combination of 100b-Digit benchmark and DISHv algorithm demonstrates how to bridge a gap between recent continuous optimizers and different discrete benchmarks. Also, vice-versa, this demonstrates how to bridge a gap between recent discrete benchmarks and different continuous optimizers. It also takes an algorithm from contains domain and demonstrates how to combine it for pligging in to the discrete domain benchmark, which might encourage applicability of search algorithm development with different domains and studying their mechanisms.

In future work, the proposed benchmark discretization and algorithm enhancements might be proposed to other benchmarks and algorithms, respectively. Hence, the discretization mechanism presented in this paper might become important for bridging the gap between discrete and continuous optimization algorithms and their applicability. By changing number of underlying continuous variables D_c and their definition intervals would also change the dimension and hence hardness of the problems, making them scalable. Another line worth investigating thoroughly is what many easy continuous problems may result in quite hard discrete problems with this encoding, especially when the optimum is not at all-zeros. Also, by using some other rule than selecting half best runs, a different benchmark could be established to assign the scores, which opens a whole new horizon to research. DISH-based algorithms might also be applied to other real world applications, like deep ocean underwater glider path planning. Additionally, as single objective benchmark problems can be transformed into dynamic, niching composition, computationally expensive, and many other classes of problems, the proposed benchmark could also be extended to such classes. Another interesting aspect of algorithm implementation on HPC systems is also evaluation of parallelization strategies over value-to-reach goals, to compute the benchmark using any HPC approach. There also exists plethora of additional search mechanisms, e.g. SQP local search [37] or mixed-integer, which might later be hybridized for special-purpose population algorithms, such as for

the new 100b-Digit Challenge domain. One important further work is also landscape analysis of the new benchmark functions.

ACKNOWLEDGMENTS

This paper is based upon work from COST Action CA15140 "Improving Applicability of Nature-Inspired Optimisation by Joining Theory and Practice (ImAppNIO)" supported by COST. This paper is also based upon work from COST Action IC1406 High-Performance Modelling and Simulation for Big Data Applications (cHiPSet), supported by COST (European Cooperation in Science and Technology). The author acknowledges the financial support from the Slovenian Research Agency (Research Core Funding No. P2-0041) and EU support under Project No. 5442-24/2017/6 (HPC – RIVR). Thanks also to the two anonymous reviewers.

REFERENCES

[1] Rawaa Dawoud Al-Dabbagh, Ferrante Neri, Norisma Idris, and Mohd Sapiyan Baba. 2018. Algorithmic design issues in adaptive differential evolution schemes: Review and taxonomy. *Swarm and Evolutionary Computation* 43 (2018), 284–311.

[2] Noor H. Awad, Mostafa Z. Ali, Ponnuthurai N. Suganthan, and Robert G. Reynolds. 2016. An Ensemble Sinusoidal Parameter Adaptation incorporated with L-SHADE for Solving CEC2014 Benchmark Problems. In *2016 IEEE World Congress on Computational Intelligence (IEEE WCCI 2016), Vancouver, Canada*. 2958–2965.

[3] Folkmar Bornemann, Dirk Laurie, Stan Wagon, and JÖrg Waldvogel. 2004. *The SIAM 100-digit challenge: a study in high-accuracy numerical computing*. Vol. 86. SIAM.

[4] Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. 2013. A survey on optimization metaheuristics. *Information Sciences* 237 (2013), 82–117.

[5] J. Brest, M. Sepesy Maučec, and B. Bošković. 2017. Single objective real-parameter optimization: algorithm jSO. In *2017 IEEE Congress on Evolutionary Computation*. 1311–1318.

[6] Swagatam Das, Sayan Maity, Bo-Yang Qu, and Ponnuthurai Nagaratnam Suganthan. 2011. Real-parameter evolutionary multimodal optimization – A survey of the state-of-the-art. *Swarm and Evolutionary Computation* 1, 2 (2011), 71–88.

[7] Swagatam Das, Sankha Subhra Mullick, and P.N. Suganthan. 2016. Recent advances in differential evolution – An updated survey. *Swarm and Evolutionary Computation* 27 (2016), 1–30.

[8] S. Das and P. N. Suganthan. 2011. Differential Evolution: A Survey of the State-of-the-art. *IEEE Transactions on Evolutionary Computation* 15, 1 (2011), 4–31.

[9] A. E. Eiben and J. E. Smith. 2003. *Introduction to Evolutionary Computing (Natural Computing Series)*. Springer.

[10] A. Glotić and A. Zamuda. 1 March 2015. Short-term combined economic and emission hydrothermal optimization by surrogate differential evolution. *Applied Energy* 141 (1 March 2015), 42–56.

[11] Shu-Mei Guo, Jason Sheng-Hong Tsai, Chin-Chang Yang, and Pang-Han Hsu. 2015. A self-optimization approach for L-SHADE incorporated with eigenvector-based crossover and successful-parent-selecting framework on CEC 2015 benchmark set. In *Evolutionary Computation (CEC), 2015 IEEE Congress on*. IEEE, 1003–1010.

[12] N. Hansen, A Auger, D. Brockhoff, D. Tušar, and T. Tušar. 2016. COCO: Performance Assessment. *ArXiv e-prints* arXiv:1605.03560 (2016).

[13] Dongli Jia, Xintao Duan, and Muhammad Khurram Khan. 2013. An Efficient Binary Differential Evolution with Parameter Adaptation. *International Journal of Computational Intelligence Systems* 6, 2 (2013), 328–336.

[14] Tao Li, Hongbin Dong, and Jing Sun. 2019. Binary differential evolution based on individual entropy for feature subset optimization. *IEEE Access* (2019).

[15] Seyedali Mirjalili and Andrew Lewis. 2013. S-shaped versus V-shaped transfer functions for binary particle swarm optimization. *Swarm and Evolutionary Computation* 9 (2013), 1–14.

[16] F. Neri and V. Tirronen. 2010. Recent Advances in Differential Evolution: A Survey and Experimental Analysis. *Artificial Intelligence Review* 33, 1–2 (2010), 61–106.

[17] Microprocessor Standards Committee of the IEEE Computer Society. 2008. IEEE Std 754™-2008 (Revision of IEEE Std 754-1985), IEEE Standard for Floating-Point Arithmetic. *Standards* (2008), 1–70. https://doi.org/10.1109/IEEESTD.2008.4610935

[18] Karol R Opara and Jarosław Arabas. 2019. Differential Evolution: A survey of theoretical analyses. *Swarm and evolutionary computation* 44 (2019), 546–558.

[19] Gary Pampara, Andries Petrus Engelbrecht, and Nelis Franken. 2006. Binary differential evolution. In *2006 IEEE International Conference on Evolutionary Computation*. IEEE, 1873–1879.

[20] Raghav Prasad Parouha and Kedar Nath Das. 2016. DPD: An intelligent parallel hybrid algorithm for economic load dispatch problems with various practical constraints. *Expert Systems with Applications* 63 (2016), 295–309.

[21] Adam P Piotrowski. 2017. Review of Differential Evolution population size. *Swarm and Evolutionary Computation* 32 (2017), 1–24.

[22] Adam P Piotrowski and Jaroslaw J Napiorkowski. 2018. Step-by-step improvement of JADE and SHADE-based algorithms: Success or failure? *Swarm and evolutionary computation* 43 (2018), 88–108.

[23] K. V. Price, N. H. Awad, M. Z. Ali, and P. N. Suganthan. 2018. *The 100-Digit Challenge: Problem Definitions and Evaluation Criteria for the 100-Digit Challenge Special Session and Competition on Single Objective Numerical Optimization*. Technical Report, Nanyang Technological University, Singapore, November 2018. Vacaville, California, USA and School of EEE, Nanyang Technological University, Singapore and School of Computer Information Systems, Jordan University of Science and Technology, Jordan.

[24] Kenneth Sörensen. 2015. Metaheuristics—the metaphor exposed. *International Transactions in Operational Research* 22, 1 (2015), 3–18.

[25] R. Storn and K. Price. 1997. Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization* 11 (1997), 341–359.

[26] Ryo Tanabe and Akira Fukunaga. 2013. Evaluating the performance of SHADE on CEC 2013 benchmark problems. In *2013 IEEE Congress on Evolutionary Computation*. IEEE, 1952–1959.

[27] Ryoji Tanabe and Alex Fukunaga. Date of Publication 25 January 2019 (in press). DOI: 10.1109/TCYB.2019.2892735. Reviewing and Benchmarking Parameter Control Methods in Differential Evolution. *IEEE Transactions on Cybernetics* (Date of Publication 25 January 2019 (in press). DOI: 10.1109/TCYB.2019.2892735).

[28] Ryoji Tanabe and Alex S Fukunaga. 2014. Improving the search performance of SHADE using linear population size reduction. In *2014 IEEE Congress on Evolutionary Computation*. IEEE, 1658–1665.

[29] Lloyd N Trefethen. 2002. The $100, 100-Digit Challenge. *SIAM News* 35 (2002), 1–3.

[30] Adam Viktorin, Roman Senkerik, Michal Pluhacek, Tomas Kadavy, and Aleš Zamuda. Available online 12 November 2018. Distance Based Parameter Adaptation for Success-History based Differential Evolution. *Swarm and Evolutionary Computation* (Available online 12 November 2018). https://doi.org/10.1016/j.swevo.2018.10.013

[31] A. Zamuda. 2019. Function Evaluations Upto 1e+12 and Large Population Sizes Assessed in Distance-based Success History Differential Evolution for 100-Digit Challenge and Numerical Optimization Scenarios (DISH-chain1e+12): A competition entry for "100-Digit Challenge, and Four Other Numerical Optimization Competitions" at The Genetic and Evolutionary Computation Conference (GECCO) 2019. In *Genetic and Evolutionary Computation Conference Companion (GECCO âĂŹ19 Companion), July 13–17, 2019, Prague, Czech Republic*. ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3319619.3326751.

[32] A. Zamuda. April 2016. Differential Evolution and Large-Scale Optimization Applications. *IGI Global, InfoSci-Videos* (April 2016). https://doi.org/10.4018/978-1-5225-0729-1

[33] A. Zamuda and J. Brest. 2012. Population Reduction Differential Evolution with Multiple Mutation Strategies in Real World Industry Challenges. In *Swarm and Evolutionary Computation (Lecture Notes in Computer Science)*, Leszek Rutkowski, Marcin Korytkowski, Rafal Scherer, Ryszard Tadeusiewicz, Lotfi Zadeh, and Jacek Zurada (Eds.). Springer, 154–161.

[34] A. Zamuda and J. Brest. 2014. Vectorized procedural models for animated trees reconstruction using differential evolution. *Information Sciences* 278 (2014), 1–21.

[35] A. Zamuda and J. Brest. 2015. Self-adaptive control parameters' randomization frequency and propagations in differential evolution. *Swarm and Evolutionary Computation* 25 (2015), 72–99.

[36] Aleš Zamuda and Janez Brest. 2018. On Tenfold Execution Time in Real World Optimization Problems with Differential Evolution in Perspective of Algorithm Design. In *2018 25th International Conference on Systems, Signals and Image Processing (IWSSIP)*. IEEE, 1–5.

[37] A. Zamuda, J. Brest, B. Bošković, and V. Žumer. 2009. Differential Evolution with Self-adaptation and Local Search for Constrained Multiobjective Optimization. In *IEEE Congress on Evolutionary Computation 2009*. IEEE Press, 195–202.

[38] Aleš Zamuda and José Daniel Hernández Sosa. 2019. Success history applied to expert system for underwater glider path planning using differential evolution. *Expert Systems with Applications* 119, 1 April 2019 (2019), 155–170.

[39] A. Zamuda, J. D. Hernández Sosa, and L. Adler. 2016. Constrained Differential Evolution Optimization for Underwater Glider Path Planning in Sub-mesoscale Eddy Sampling. *Applied Soft Computing* 42 (2016), 93–118.