Hybrid Henry gas solubility optimization algorithm with dynamic cluster-to-algorithm mapping

Kamal Z. Zamli · Md. Abdul Kader · Saiful Azad · Bestoun S. Ahmed

Received: 29 May 2020 / Accepted: 11 December 2020 / Published online: 19 January 2021
© The Author(s), under exclusive licence to Springer-Verlag London Ltd. part of Springer Nature 2021

Abstract
This paper discusses a new variant of Henry Gas Solubility Optimization (HGSO) Algorithm, called Hybrid HGSO (HHGSO). Unlike its predecessor, HHGSO allows multiple clusters serving different individual meta-heuristic algorithms (i.e., with its own defined parameters and local best) to coexist within the same population. Exploiting the dynamic cluster-to-algorithm mapping via penalized and reward model with adaptive switching factor, HHGSO offers a novel approach for meta-heuristic hybridization consisting of Jaya Algorithm, Sooty Tern Optimization Algorithm, Butterfly Optimization Algorithm, and Owl Search Algorithm, respectively. The acquired results from the selected two case studies (i.e., involving team formation problem and combinatorial test suite generation) indicate that the hybridization has notably improved the performance of HGSO and gives superior performance against other competing meta-heuristic and hyper-heuristic algorithms.

Keywords Hybrid meta-heuristic algorithm · Henry Gas Solubility Optimization Algorithm · Search-based Software Engineering

1 Introduction
The meta-heuristic algorithm can be seen as a template for solving general optimization problems. Guided by a given (minimization or maximization) objective function, every meta-heuristic algorithm provides a specific mechanism to explore (i.e., roaming the new potential region in the search space for better solution alternatives) and to exploit (i.e., manipulating the search space in the vicinity of the known best) the given search space efficiently. To ensure good performance, the exploration and exploitation need to be properly balanced during the actual search process (i.e., taking into consideration the search space contour). Given the diverse source of inspirations (e.g., physical-based [1], swam-based [2], natural evolution-based [3]) for each meta-heuristic algorithm, the exploration and exploitation process in each meta-heuristic algorithm can be significantly different.

To date, the best classification of existing meta-heuristic algorithms with clear group demarcation relates to a single-solution and population-based algorithms. As the name suggests, single-solution-based meta-heuristic algorithms uses a single-solution model. In each iteration, the same solution is updated until the iteration ends. The advantage of a single-solution-based model is that it is memoryless. As a result, the execution overhead of single-solution-based algorithms is often low. However, single-solution-based algorithms often have slow convergence as the algorithms rely on a single point exploration. On the same note, the performance of single-solution-based meta-heuristic algorithms is also sensitive to its initial search point. Some examples of single-solution-based meta-
heuristic algorithms include Simulated Annealing (SA) [4], Guided Local Search (GLS) [5], Variable Neighborhood Algorithm (VNS) [6], Threshold Accepting Method (TA) [7], and Tabu Search (TS) [8].

On the other hand, population-based meta-heuristic algorithms exploit a set of solution candidates (i.e., memory) via its population. Throughout the search process, every candidate solution in the population is updated until the iteration ends. The main advantage of the population-based meta-heuristic algorithm over the single population-based counterpart relates to exploration. Given that each population may be residing in a different region of the search space, convergence of population-based is much faster than single population-based meta-heuristic algorithms. However, the execution overhead of population-based meta-heuristic algorithms is often higher. Some examples of population-based meta-heuristic algorithms include Jaya Algorithm (JA) [9], Sooty Tern Optimization Algorithm (STOA) [10], Butterfly Optimization Algorithm (BOA) [11], Owl Search Algorithm (OSA) [12], Henry Gas Solubility Optimization Algorithm (HGSO) [13], Manta-Ray Foraging Optimization (MRFO) [14], Water Strider Algorithm (WSA) [15], Artificial Electric Field Algorithm (AEFA) [16], Equilibrium Optimizer (EO) [17], Side-Blotched Lizard Algorithm (SLA) [18], Political Optimizer (PO) [19], Poor and Rich Optimization Algorithm (PROA) [20], and Tunicate Swarm Algorithm (TSA) [21].

Henry Gas Solubility Optimization Algorithm (HGSO) is a recently developed population-based meta-heuristic algorithm in the literature. The notable feature of HGSO that stands out against other competing meta-heuristics is that the whole population (e.g., pop [0] till pop [k] in Fig. 1) is divided into a set of N clusters. Each cluster is then mapped to an independent HGSO with its parameter controls as well as its own local best. At a glance, having more than one independent HGSO in many sub-clusters can boost exploration. However, a closer look reveals two main limitations. Firstly, the HGSO-to-cluster is statically mapped (i.e., at any instance of iteration, the same mapping is used). Here, the static mapping does not consider the adaptive performance of each HGSO-to-cluster mapping for the subsequent iteration. Secondly, the HGSO-to-cluster mapping also ignores the opportunity for hybridization with other meta-heuristics. With hybridization, one can compensate the limitation of a host algorithm with the strength of others. Best results in the literature have often been associated with hybridization [22–24]. Rather than having one type of meta-heuristic-to-cluster mapping, many types of meta-heuristic-to-cluster mapping can be introduced by using more than one algorithm. Additionally, the meta-heuristic-to-cluster mapping can be made adaptive and dynamic based on the need for the current search process.

Given the aforementioned prospects, this paper proposes the development of Hybrid HGSO (HHGSO). The main contribution of this work relates to the introduction of a new HGSO variant, called HHGSO, exploiting four recently developed meta-heuristic algorithms (apart from itself as host) including Jaya Algorithm (JA) [9], Sooty Tern Optimization Algorithm (STOA) [10], Butterfly Optimization Algorithm (BOA) [11] and Owl Search Algorithm (OSA) [12]. Here, the individual mapping of the algorithm to the cluster is designed based on penalized and reward model with adaptive switching factor. Extensive performance comparison of HHGSO is undertaken considering other state-of-the-art algorithms (e.g., the original HGSO, Jaya, Sooty Tern Optimization Algorithm (STOA), Butterfly Optimization Algorithm (BOA) and Owl Search Algorithm (OSA) as well as recently developed hyper-heuristic algorithms (e.g., Exponential Monte Carlo with Counter [25, 26], modified choice function [27, 28], Improvement Selection Rules [29], and Fuzzy Inference Selection [30]) for two Search-based Software Engineering problems involving team formation problem and combinatorial test suite generation.

The paper is organized as follows. Section 2 describes the relevant related works while Sect. 3 discusses the hybridization of meta-heuristic algorithms. Section 4 presents the original HGSO. Section 5 elaborates on the proposed HHGSO. Section 6 presents our research questions as well as the experimental setup. Section 7 discusses our empirical evaluation related to all the research questions. Section 8 offers some reflection on the work undertaken. Finally, Sect. 9 gives our concluding remarks along with the scope for future work.

### 2 Relevant related works

Meta-heuristic algorithms often adopt some natural phenomena as their source of inspiration in order to search for an optimal solution. Among the recent meta-heuristic algorithms that are relevant to the current work are Jaya [9], Sooty Tern Optimization Algorithm (STOA) [10], Butterfly Optimization Algorithm (BOA) [11], and Owl Search Algorithm (OSA) [31]. More detailed descriptions of these aforementioned algorithms in terms of their mathematical models are outlined in Sect. 5.

Jaya algorithm is a population-based meta-heuristic algorithm proposed by Rao [9]. The notable feature of Jaya is the fact that it is a parameter-free algorithm that does not rely on any parameter controls making it a straightforward algorithm for adoption. Nonetheless, not having any parameter controls can also be a curse to Jaya as effective control of exploration and exploitation can be problematic.
Sooty Tern Optimization Algorithm (STOA) is also a population-based meta-heuristic algorithm. Proposed by Dhiman and Kaur [10], STOA mimics the migration and attacking behaviors of the sea bird sooty tern in nature as its source of inspiration. Using the sooty tern’s flapping mode of flight to locate the prey, STOA achieves its exploitation. Meanwhile, the flocking and migrating of sooty tern to new places serve as its exploration. Owing to its rather complex model of inspiration, STOA’s adoption can be more challenging than that of Jaya.

Butterfly Optimization Algorithm (BOA) is a population-based meta-heuristic algorithm outlined by Arora et al. [11] based on the foraging and the social behavior of butterflies. The framework of BOA is based on the fragrance emitted by the butterflies, which helps other butterflies to search for food as well as a mating partner. As the fragrance is the function of sensory stimulus, setting the effective value of its sensory modality parameter for exploration and exploitation can be an issue. In fact, this sensory modality parameter is sensitive to the problem at hand and requires extensive tuning.

Meanwhile, Owl Search Algorithm (OSA) is a population-based meta-heuristic algorithm that mimics the hunting mechanism of the owls in the dark. Developed by Jain et al. [12], OSA exploits the sound intensity factor as the multiplier to explore (i.e., updating the current solution with large displacement) and exploit (i.e., updating the current solution with small displacement). As the sound intensity factor is a ratio of differences in current fitness with the worst fitness over the best fitness with the worst fitness, the exploration process can be problematic when reaching convergence. In this case, the sound intensity factor will be a very large number (i.e., close to $\infty$) as the differences between the best fitness and the worst fitness is small. In this manner, OSA is often not suitable for optimization problems with multiple peaks that require significant local search capabilities.

Given so many existing works, some might question the need to develop new (or enhanced existing) meta-heuristic algorithms. The answer can be found in the No Free Lunch Theorem of Optimization [32], which stipulates that there is no best algorithm for all optimization problems. In fact, it is this theorem that motivates the current work.

3 Related work on meta-heuristic hybridization

Without any knowledge about the nature of the optimization problem at hand, combining the strengths of different meta-heuristic algorithms (as well as compensating the limitation of host algorithm), belonging to different classes of implementations, may increase the probability of success of the overall search process. Specifically, meta-
heuristic hybridization involves combining or grouping two or more meta-heuristic algorithms to form a new hybrid algorithm. It should be noted that we differentiate hybrid algorithms from ensemble algorithms. Ensemble algorithms involve combining one or more meta-heuristic algorithms with other general artificial-based algorithms such as artificial neural network, fuzzy logic, k-mean clustering algorithm, and support vector machine to name a few.

To put meta-heuristic hybridization into perspective, consider the different possible groupings for utilizing and combining meta-heuristic algorithms. Owing to its popularity and in line with the scope of the current work, our grouping focuses only on the hybridization of population-based meta-heuristic algorithms (see Fig. 2). The first two (non-hybrid) groups, shown for completeness, are single algorithm single population (SASP) and single algorithm many populations (SAMP) class. These two classes can be ignored from our discussion as we want to focus on hybridization methodologies. The two remaining groups are many algorithms single population (MASP) and many algorithms many populations (MAMP). These two groupings can further be decomposed into low-level hybrids and high-level hybrids based on Talbi [33] and Zamli [24]. Low-level hybrids require changes in the internal structure of the source code to alter certain function(s) of a host algorithm with other meta-heuristic algorithms. Meanwhile, high-level hybrids combine two or more meta-heuristic algorithms as black-box components (i.e., the internals are non-intersecting with the implementation of the host algorithm).

Complementing the work by Talbi [33], the two given grouping of MASP and MAMP can be elaborated in further into as MASP Low-Level Hybrid (MASP-LLH), MASP High-Level Hybrid (MASP-HLH), MAMP Low-Level Hybrid (MAMP-LLH), and MAMP High-Level Hybrid (MAMP-HLH), respectively.

Concerning MASP-LLH, the implementation can be in the form of a relay (i.e., each meta-heuristic algorithm is sequentially applied) or cooperative (with no given ordering) applied to the same population. The advantage of MASP-LLH is that the user has direct control of the structure of whole combined algorithms. The limitation of MASP-LLH is that the developed hybrid can be too problem-specific.

Examples of relay MASP-LLH include the work of Nasser et al. [34], Rambabu et al. [35], and Long et al. [36], respectively. The work of Nasser et al. [34] implements Flower Pollination Algorithm (FPA) [37] with the mutation operator. The work also adopts an elitism operator to improve the current poor solutions. Meanwhile, Rambabu et al. [35] integrate Artificial Bee Colony (ABC) [38] with the Monarch Butterfly Optimization Algorithm (MBOA) [39]. Specifically, the MBOA is integrated as part of the employer bee adjusting phase preceding the onlooker bee phase. Long et al. [36] developed a hybrid algorithm by integrating Grey Wolf Optimizer (GWO) [40] with Cuckoo Search (CS) [41] for parameter extraction of solar photovoltaic models. In the work, the update of a new grey wolves’ location is achieved via the CS levy flight operator.

Meanwhile, examples of cooperative MASP-LLH are the work of Zamli et al. [23], Alotaibi [42], and Sharma et al. [43], respectively. Zamli et al. [23] develop a cooperative hybrid algorithm that allows low-level selection between the sine operator and the cosine operator from Sine–Cosine Algorithm (SCA) [44], levy flight operator from Cuckoo Search Algorithm (CSA) [41] and crossover operator from Genetic Algorithm (GA) [45] using the Q-learning framework. Alotaibi [42] develops a low-level hybrid that integrates the Firefly Algorithm (FA) [46] with Jaya Algorithm (JA) [9] for video copyright protection. The selection of either FA or Jaya update is achieved based on the predefined trial constant. Initially, the algorithm uses FA while after reaching the trial constant, Jaya will take over. In other work, Sharma et al. [43] develop an ensemble of Butterfly Optimization Algorithm (BOA) [11] and Symbiosis Organisms Search (SOS) [47] where the global search ability of BOA and the local search ability of

![Population-based meta-heuristic algorithm implementations](image-url)
SOS are combined for solving the general global optimization problem.

As far as MASP-MLH is concerned, the implementation can also be in the form of a relay (i.e., each meta-heuristic algorithm is sequentially applied) or cooperative (with no given ordering) applied to the same population. MASP-MLH can often be associated with hyper-heuristic algorithms (termed as (meta)-heuristic to choose (meta)-heuristic) \([48–51]\). By using many (meta)-heuristic algorithms (or their associated search operators), the hyper-heuristic methodology can be considered as a form of hybridization. Focusing on the generalized method for solving optimization problems (i.e., owing to strict separation from implementation and problem domain), hyper-heuristic algorithms come in two flavors, namely generative hyper-heuristic algorithms and selective hyper-heuristic algorithms. The generative hyper-heuristic algorithm can customize its combination of new heuristics from a pool of possible heuristics. In contrast, the selective hyper-heuristic algorithms select the heuristics from a predefined set of heuristics. In essence, unlike selective hyper-heuristic algorithms, generative hyper-heuristic algorithms lend themselves to different types of optimization problems with minimal changes.

MASP-MLH grouping, however, is not exclusively mapped to only hyper-heuristic algorithms. The grouping can also include other forms of many algorithms’ hybridization as long as each participating algorithm is treated as individual black-box components and with non-intersecting feature replacement of the host algorithm.

Examples of relay MASP-MLH implementation are the work of Luan et al. \([52]\), Noori and Ghannadpour \([53]\), and Lepagnot et al. \([54]\), respectively. Luan et al. \([52]\) hybridized the Genetic Algorithm (GA) \([45]\) with Ant Colony Optimization (ACO) \([55]\) to settle the supplier selection problem. This high-level hybrid improves the GA and ACO separately to enhance its efficiency and effectiveness. The work by Noori and Ghannadpour \([53]\) adopts three levels of the high-level relay optimization process. The Genetic Algorithm (GA) \([45]\) serves as the main optimization algorithm and Tabu Search (TS) \([8]\) as an improvement method. In each level, heuristics incorporate local exploitation in the evolutionary search in order to solve the Multi-Depot Vehicle Routing Problem with Time Windows. In other work, Lepagnot et al. \([54]\) propose a high-level relay hybrid algorithm that combines the Multiple Local Search Algorithm (MLSA) \([56]\) for dynamic optimization, the Success History-Based Adaptive Differential Evolution (SHADE) \([57]\), and the Standard Particle Swarm Optimization (PSO) \([58]\). The hybrid algorithm is then subjected to a selected benchmark black-box optimization problem.

As for cooperative MASP-MLH, the work of Ahmad et al. \([49]\), and Zamli et al. \([29,50]\) can be highlighted as relevant examples. Ahmad et al. \([49]\) propose a Monte Carlo-based hyper-heuristic technique that embeds the Q-learning framework as an adaptive meta-heuristic selection and acceptance mechanism. The work adopts low-level search operations from the Cuckoo Search Algorithm (CSA) \([41]\), Jaya Algorithm (JA) \([9]\), and Flower Pollination Algorithm (FPA) \([9]\). In similar work, Zamli et al. \([29]\) develop Tabu Search \([8]\)-based hyper-heuristic algorithm, which rides on Teaching Learning-based Optimization Algorithm (TLBO) \([59]\), CSA \([41]\), and PSO \([58]\). As an extension of the work in \([29]\), Zamli et al. \([50]\) integrate the Mamdani Fuzzy Inference Selection with its hyper-heuristic algorithm along with low-level search operations based on the FPA \([9]\), JA \([9]\), GA crossover \([45]\) and TLBO \([59]\).

Concerning MAMP-LLH, the implementation typically lends itself toward parallel execution. Relay-based MAMP-LLH is possible but often ignored as the approach does not promote parallelism. Cooperative-based MAMP-LLH is preferable, but there is potential overhead related to parallelism in terms of the need to coordinate and synchronize the contribution from each algorithm’s population improvement. Being low-level, cooperative MAMP-LLH can also be too problem-specific (e.g., integrating domain-specific assumptions into the developed hybrid).

Examples of cooperative MAMP-LLH include the work of Pourvaziri and Naderi \([60]\), Zhou and Yao \([61]\), and Chen et al. \([62]\), respectively. Pourvaziri and Naderi \([60]\) introduce a hybrid multi-population genetic algorithm for the dynamic facility layout problem, which adopts the local search mechanism from Simulated Annealing (SA) \([4]\). Zhou and Yao \([61]\) develop a multi-population parallel self-adaptive differential Artificial Bee Colony (ABC) \([38]\) algorithm, where the distinct hybrid evolutionary operators borrowed from the Differential Evolution (DE) \([63]\) are adopted during the evolution process. Meanwhile, Chen et al. \([62]\) incorporate chaos strategy, multi-population, and DE \([63]\) as part of low-level hybrid Harris Hawks Optimization (HHO) \([64]\) implementation.

Like MAMP-LLH, MAMP-MLH also lends itself toward parallel execution but does not favor a relay-based implementation. Unlike MAMP-LLH, the interaction between meta-heuristic algorithms is often at a high-level of abstraction (e.g., through black-box parameter interface), hence promoting better generalization. Similar parallelism issues need to be addressed in terms of the overhead of coordination and synchronization of the contribution from each algorithm’s population improvement.

Examples of cooperative MAMP-MLH include the work of Zhang et al. \([22]\), Cruz-Chávez et al. \([65]\), and Łapa et al. \([66]\). Zhang et al. \([22]\) hybridized CSA \([41]\) with DE
to solve constrained engineering problems that can find satisfactory global optima and avoid premature convergence. This work divides the population into two subgroups and adopts CSA and DE for these two subgroups independently. In another work, Cruz-Chávez et al. [65] present a hybrid GA [45] with collective communication using distributed processing for the job shop scheduling problem. In this hybrid, diversification is performed by using distributed processing for the job shop scheduling for our hybridization. This highlights the HGSO implementation as the host algorithm balancing the exploration and exploitation of the search space is made through genetic approximation. Meanwhile, Łapa et al. [66] propose a hybrid multi-population-based approach where specified populations are processed using different population-based algorithms and synchronized accordingly for selecting the controller’s structure and parameters.

Summing up, all the aforementioned works suggest that hybridization is useful to enhance the search performance of the original meta-heuristic algorithm (i.e., in terms of balancing the exploration and exploitation of the search process). Taking the current work further, the next section highlights the HGSO implementation as the host algorithm for our hybridization.

4 Henry Gas Solubility Optimization Algorithm

HGSO is inspired by the solubility behavior of gases in liquids [67] based on Henry’s law [68]. Henry’s law dictates that the solubility of a gas in the given fluid is inversely proportional to the temperature and proportionate to the corresponding gas’s pressure [69]. Mathematically, Henry’s law can be expressed by Eq. (1) whereby $S_g$ corresponds to the solubility of a gas:

$$S_g = H \times P_g$$

where $H$ is Henry’s constant, and $P_g$ represents the partial pressure of the gas.

The relation between the Henry’s constant and the temperature dependence of a system can be described with the Van’t Hoff equation as follows:

$$\frac{d \ln H}{d(1/T)} = \frac{-\nabla_{sol}E}{R}$$

(2)

where $\nabla_{sol}E$ is the enthalpy of dissolution, $R$ is the gas constant, and $A$ and $B$ are two parameters for $T$, which depends on $H$.

Based on the Van’t Hoff equation [see Eq. (2)], Eq. (1) can be simplified as [see Eq. (3)]:

$$H(T) = \exp(B/T) \times A$$

(3)

where $H$ is a function of parameters $A$ and $B$, which $A$ and $B$ are two parameters for $T$ dependence of $H$. Alternatively, one can create an expression based on $H^0$ at the reference temperature $T = 298.15 \text{ K}$.

$$H(T) = H^0 \times \exp\left(\frac{-\nabla_{sol}E}{R} \left(\frac{1}{T} - \frac{1}{T^0}\right)\right)$$

As the Van’t Hoff equation is valid when $\nabla_{sol}E$ is a constant, Eq. (4) can also be expressed as follows:

$$H(T) = \exp\left(-C \times \left(\frac{1}{T} - \frac{1}{T^0}\right)\right) \times H^0$$

(5)

Based on the said law, the HGSO algorithm can be described in eight steps as follows:

**Step 1: Initialization process** The positions of $N$ gases are randomly initialized based on Eq. (6):

$$X_i(t + 1) = X_{\min} + r \times (X_{\max} - X_{\min})$$

(6)

where $t$ and $r$ represent the current iteration number and a uniform random number, respectively. The position of the $i$th gas in population $N$ is denoted by $X_{(i)}$. $X_{\max}$ represents the upper bound and $X_{\min}$ represents the lower bound of the problem.

The Henry’s constant, partial pressure, and constant for the $i$th gas and $j$th cluster are denoted as $H_{j}(t)$, $P_{ij}$, $C_i$, accordingly. The parameters are initialized by using Eq. (5):

$$H_{j}(t) = l_1 \times \text{rand}(0,1), \quad P_{ij} = l_2 \times \text{rand}(0,1), \quad C_i = l_3 \times \text{rand}(0,1)$$

(7)

where $l_1 = 5E-02$, $l_2 = 100$, and $l_3 = 1E-02$ are constants.

**Step 2: Clustering** The size of the clusters is equal to the number of type of gases. The gases in the same cluster have identical Henry’s constant value ($H_j$).

**Step 3: Evaluation** Initially, the clusters are calculated to find out the best gas in their corresponding cluster. Then, the cluster bests are ranked based on the fitness and find out the optimal gas in the entire population.

**Step 4: Update Henry’s coefficient** Henry’s coefficient $H_j$ is modified according to Eq. (8) for cluster $j$, and $t$ represents the total number of iterations:

$$H_j(t + 1) = H_j(t) \times \exp\left(-C_j \times \left(\frac{1}{T(t)} - \frac{1}{T^0}\right)\right), \quad T(t) = (-t/\text{iter})$$

(8)

**Step 5: Update solubility** The HGSO modifies the solubility of gas $i$ in cluster $j$ ($S_{ij}$) according to Eq. (9) where and $P_{ij}$ is the partial pressure on gas $i$ in cluster $j$ and $K$ is a constant:

$$S_{ij}(t) = K \times H_j(t + 1) \times P_{ij}(t)$$

(9)

**Step 6: Update position** The HGSO modifies the position of gas $i$ in cluster $j$ ($X_{(ij)}$) in iteration $t$ using Eq. (8) where $r$ is a random constant.
\[
X_{ij}(t + 1) = X_{ij}(t) + F \times r \times \gamma \times (X_{i\text{best}}(t) - X_{ij}(t)) \\
+ F \times r \times \alpha \times (S_{ij}(t) \times X_{\text{best}}(t) - X_{ij}(t))
\]
\[
\gamma = \beta \times \exp \left( - \frac{F_{\text{best}}(t) + \varepsilon}{F_{ij}(t) + \varepsilon} \right), \quad \varepsilon = 0.05
\]

(10)

\(X_{i\text{best}}\) and \(X_{(i,\text{best})}\) represented the best of the swarm and cluster, respectively, which are directly responsible for controlling the exploration and exploitation stages. Additionally, \(\alpha, \beta, \) and \(\gamma\) represent the influence of other gases on current gas, a random constant, and the interaction ability of gases in the same cluster. The fitness of gas \(i\) in cluster \(j\) is denoted by \(F_{(i,j)}\). In contrast, the fitness of the global best is symbolized by \(F_{\text{best}}\). To ensure diversity, the flag \(F\) controls the search direction.

**Step 7: Escape from local optimum** This step ranks and selects the number of worst agents (\(N_w\)) using Eq. (11) to escape from local optimum where \(N\) is the number of search agents:

\[
N_w = N \times (\text{rand}(c_2 - c_1) + c_1), \quad c_1 = 0.1, \quad c_2 = 0.2
\]

(11)

**Step 8: Update the position of the worst agents** The position update is given by Eq. (12).

\[
X_{(i,j)} = X_{\text{min}(i,j)} + r \times (X_{\text{max}(i,j)} - X_{\text{min}(i,j)})
\]

(12)

where \(r, X_{\text{min}},\) and \(X_{\text{max}}\) represent a random number, lower bound, and upper bound of the problem, respectively.

The pseudocode of HGSO is summarized in Fig. 3.

5 The proposed hybrid HGSO

The proposed Hybrid HGSO (HHGSO) can be visualized, as in Fig. 4. At any iteration, HHGSO provides dynamic meta-heuristics-to-cluster mapping. For instance, as iteration = 0, the meta-heuristic-to-cluster mapping could be different from the one at iteration = \(J\) or iteration = \(K\). The dynamic meta-heuristic-to-cluster mapping is achieved through penalized and reward model with adaptive switching factor.

5.1 Penalized and reward model with adaptive switching factor

In a nutshell, the penalized and reward model considers the cooperative performance of a particular meta-heuristic-to-cluster mapping as a whole. If the cooperative performance at iteration = \(K\) realizes an improvement of the population’s global best, then the meta-heuristic-to-cluster mapping will be rewarded and be maintained for the next iteration = \(K + 1\). Otherwise, the meta-heuristic-to-cluster mapping will be updated randomly based on adaptive switching factor SF as follows [see Eq. (13)]:

\[
SF = SF_{\text{max}} + \frac{t(SF_{\text{min}} - SF_{\text{max}})}{\text{Maxiteration}}
\]

(13)

where the minimum switching factor \(SF_{\text{min}} = 0.2\) and the maximum switching factor \(SF = 0.8\). \(t\) is the current iteration and \(\text{Maxiteration}\) is the maximum iteration.

A random probability \(p\) will be generated after each complete iteration and be compared with SF in order to reward or to penalize the current meta-heuristic-to-cluster mapping. In the early part of the search iteration, the switching factor SF will be large meaning that the meta-heuristic-to-cluster mapping is likely to change whenever the performance of the overall search is poor. In this manner, each cluster can have a different update operator from more than one meta-heuristic algorithm to ensure sufficient exploration. Toward the end during convergence, the switching SF becomes small, minimizing any significant change to the meta-heuristic-to-cluster mapping supporting better exploitation.

5.2 Constituent meta-heuristic algorithms

Apart from HGSO itself as host, HHGSO leverages four other meta-heuristic algorithms as its hybridization vehicle (i.e., Jaya, Sooty Tern Optimization Algorithm (STOA), Butterfly Optimization Algorithm (BOA), and Owl Search Algorithm (OSA)). The brief description of each meta-heuristic algorithm is presented in the following subsections.

5.2.1 Jaya algorithm (JA)

Jaya is a parameter-free meta-heuristic algorithm developed by Rao [9]. Jaya works by establishing the solution to problems through avoiding the worst solutions and moving toward the best optimal solution. Jaya modifies its solutions based on the best and worst solutions using Eq. (14).

\[
X_i(t + 1) = X_i(t) + r_1(X_{\text{best}} - |X_i(t)|) - r_2(X_{\text{worst}} - |X_i(t)|)
\]

(14)

where \(X_{\text{best}}\) is the value of the variable for \(F_{\text{best}}\) and \(X_{\text{worst}}\) is the value of the variable for \(F_{\text{worst}}\) and \(X_{i}(t + 1)\) as the updated \(i^{th}\) value of \(X_{i}(t)\). Here, \(r_1\) and \(r_2\) are the two random scaling factors in the range \([0, 1]\).
[represented in Eq. (15) until Eq. (19)] and attacking behavior [represented in Eq. (20) until Eq. (24)] of sooty terns in STOA are mathematically simulated as follows:

\[ C_i = S_A \times X_i(t) \]  \hspace{1cm} (15)

where \( C_i \) is the position of search agent, \( X_i(t) \) represents the current position of search agent at iteration \( t \), \( S_A \) indicates the movement of the search agent in a given search
space. $C_f$ is a controlling variable to adjust the $S_A$ which is linearly decreased from $C_f$ to 0. $t$ is the current iteration ($t = 0, 1, \ldots \text{Max}_{\text{iteration}}$) and $\text{Max}_{\text{iteration}}$ is the maximum number of iterations.

$$M_t = C_B \times (X_{\text{best}}(t) - X_i(t))$$  \hspace{1cm} (17)

$$C_B = 0.5 \times \text{rand}(0, 1)$$  \hspace{1cm} (18)

where $M_t$ represents the different locations of the search agent $X_i(t)$ toward the best fittest search agent $X_{\text{best}}(t)$. $C_B$ is a uniformly distributed random variable (responsible for better exploration).

$$D_t = C_i \times M_t$$  \hspace{1cm} (19)

where $D_t$ defines the gap between the search agent and best fittest search agent

$$x' = R_{\text{radius}} \times \sin(m)$$  \hspace{1cm} (20)

$$y' = R_{\text{radius}} \times \cos(m)$$  \hspace{1cm} (21)

$$z' = R_{\text{radius}} \times m$$  \hspace{1cm} (22)

$$r = u \times e^v$$  \hspace{1cm} (23)

where $R_{\text{radius}}$ represents the radius of each turn of the spiral, $m$ represents the variable lies between the range of $[0 \leq k \leq 2\pi]$, $u$ and $v$ are constants to define the spiral shape, and $e$ is the base of the natural logarithm. The candidate solution update is given in Eq. (24) as follows:

$$X_i(t) = (D_t(t) \times (x' + y' + z')) \times X_{\text{best}}(t)$$  \hspace{1cm} (24)

where $X_{\text{best}}(t)$ is the best fittest search agent. $D_t(t)$ defines the gap between the search agent and best fittest search agent. $x'$, $y'$, and $z'$ represent the angle of attack.

### 5.2.3 Butterfly Optimization Algorithm (BOA)

Butterfly Optimization Algorithm (BOA) is a population-based meta-heuristic algorithm proposed by Arora et al. [11]. The BOA mimics the foraging and the social behavior of the butterflies. In a nutshell, butterflies use their senses for finding foods, searching for a mating partner, migrating from one place to another, and escaping from enemies.

BOA exploits the fragrance as a function of the physical intensity stimulus as part of its movement based on Eq. (25).

$$f_i = cI^p$$  \hspace{1cm} (25)

where $c = 0.01$ is a sensory modality, $I$ is the stimulus intensity typically upper–lower bound, $a$ is power exponent linearly updated from 0.1 to 0.2.

The global candidate solution update is given by Eq. (26).

$$X_i(t + 1) = X_i(t) + (r^2 \times X_{\text{best}} - X_i(t)) \times f_i$$  \hspace{1cm} (26)

where $X_i(t)$ is the solution vector $x_i$ for $i$th butterfly in iteration $t$. $X_{\text{best}}$ represents the current best solution found among all the solutions in the current iteration. $f_i$ represents the fragrance of $i$th butterfly, and $r$ is a random number between [0, 1].

Complementing the global candidate solution update, BOA as defines local candidate update as in Eq. (27).

$$X_i(t + 1) = X_i(t) + (r^2 \times X_i(t) - X_k(t)) \times f_i$$  \hspace{1cm} (27)

where $X_i(t)$ and $X_k(t)$ are $i$th and $k$th butterflies from the solution space.

### 5.2.4 Owl Search Algorithm (OSA)

Owl Search Algorithm (OSA) is a population-based meta-heuristic algorithm developed by Jain et al. [12]. OSA mimics the hunting mechanism of the owls in the dark. Like other population-based meta-heuristics, OSA stores the initial positions and fitness values of all owls in a two-dimensional matrix (i.e., $X$ and $F$). The size of each matrix is $n \times d$ where $n$ is the number of owls and $d$ represents the dimension of search space. The fitness value of the owl’s position directly relates to the sound intensity. The sound intensity of $i$th owl can be calculated by Eq. (28).

$$I_i = \frac{F_i - w}{b - w}$$  \hspace{1cm} (28)

where $F_i$ is the fitness of $i$th owl, $w$ minimum fitness, and $b$ represents the maximum fitness.

Equation (29) represents the distance information of each owl and prey. The change in intensity of the $i$th owl can be computed using Eq. (30).

$$R_i = ||X_i - V||_2$$  \hspace{1cm} (29)

$$I_{ci} = \frac{I_i}{R_i^2} + \text{rand}(0, 1)$$  \hspace{1cm} (30)

where $V$ is the location of prey calculated from fittest owl, $R_i$ is the distance of prey from the owl $X_i$.

The owls candidate solution update is given in Eq. (31).

$$X_i(t + 1) = \begin{cases} X_i(t) + \beta \times I_{ci} \times |xV - X_i(t)|, & \text{if } p_{vm} < 0.5 \\ X_i(t) - \beta \times I_{ci} \times |xV - X_i(t)|, & \text{if } p_{vm} < 0.5 \\ \end{cases}$$  \hspace{1cm} (31)

where $p_{vm}$ is the probability of vole movement, $x$ is a uniformly distributed random number in the range [0, 0.5], and $\beta$ is a linearly decreasing constant from 1.9 to 0. $I_{ci}$ is the change in intensity for $i$th owl. $V$ is the location of prey, which is achieved by the fittest owl (i.e., $X_{\text{best}}$).
5.3 HHGSO implementation

Taking into consideration the penalized and reward model with adaptive switching factor as well as all the constituent meta-heuristic algorithms, HHGSO pseudocode can be summarized in Fig. 5.

Referring to Fig. 5, the HHGSO algorithm starts with defining the algorithm list, cluster N_size, each meta-heuristic algorithm’s parameter initialization along with population initialization (see line 1 until line 5). The main iteration loop starts in line 6. The algorithm-to-cluster mapping is tracked by the division index variable (see line 7). Modulo division is used to ensure that the division index is mapped correctly to the corresponding algorithm in use at any particular iteration. The current running algorithm is mapped to the algorithm list as determined by the division index variable (see line 10 and line 13). The selection of the running meta-heuristic algorithm follows accordingly (refer to line 15 until 34). To go out of local optima, HHGSO will update N_size, poor solution with the new random ones (see line 36). Each cluster maintains its own local cluster best X_cluster_best and the selection of the global best X_best will be made among the local cluster best (see line 37). If the overall cooperative performance improves the same meta-heuristic-to-cluster is maintained. Otherwise, with decreasing adaptive switching factor, the ordering of algorithm list will be reshuffled to generate a new meta-heuristic-to-cluster mapping (line 40 until line 43). If there are more clusters than the list of defined algorithms, leftover clusters will be mapped to HGSO as the host algorithm (line 42). The iteration will stop when Maxiteration or maximum fitness evaluation is reached (line 46). In the end, the global best result X_best will be returned upon completion (refer to line 48).

6 Empirical evaluation

We have subjected our work under intensive evaluation. Our goals of the evaluation experiments are fourfold: (1) to characterize the performance of HHGSO against the original HGSO algorithm and its constituent meta-heuristic algorithms based on the benchmark team formation problem; (2) to benchmark HHGSO against hyper-heuristic algorithms (as cooperative MASP-HLH implementation) based on the benchmark combinatorial t-way test data generation; (3) to assess the effect of cluster size on the meta-heuristic-algorithm-to-cluster mapping of HHGSO; and (4) to determine the suitable combinations of minimum and maximum switch factors for HHGSO given the variations of population size.

In line with the aforementioned goals, we focus on answering the following research questions:

- **RQ1**: How is the performance of HHGSO compared to that of the original HGSO and its participating constituent algorithms?
- **RQ2**: What is the effect of cluster size with the meta-heuristic-algorithm-to-cluster mapping?
- **RQ3**: What is the suitable minimum switch factor (SF_{min}) and maximum switch factor (SF_{max}) for HHGSO with varying population size?
- **RQ4**: How good is the performance of HHGSO against its cooperative MASP-HLH counterparts (i.e., hyper-heuristic algorithms)?
- **RQ5**: Is there any overhead in terms of time performance penalty of HHGSO implementation as compared to its constituent algorithms?
- **RQ6**: How generalized can HHGSO implementation be for solving general optimization problems?

6.1 Experimental benchmark setup

We adopt an environment consisting of a machine running Windows 10, with a 2.9 GHz Intel Core i5 CPU, 16 GB 1867 MHz DDR3 RAM, and 512 GB flash storage throughout all our experiments. We implement our HHGSO in the Java programming language.

To ensure fairness, we have adopted different settings on population size, maximum iteration as well as maximum fitness evaluation for relevant experiments and their related research questions. For RQ1, we have adopted the population size N = 50 with the maximum iteration Max_iter = 100. Here, we limit the maximum fitness evaluation Max_{f_eval} = 2500.

Meanwhile, for RQ3, we vary the population size from 15, 25, 50, 75, and 100. At the same time, we also vary SF_{min} and SF_{max} from 5 different combinations of \{SF_{max} = 0.1, SF_{min} = 0.9\}, \{SF_{max} = 0.2, SF_{min} = 0.8\}, \{SF_{max} = 0.3, SF_{min} = 0.7\}, \{SF_{max} = 0.4, SF_{min} = 0.6\}, and \{SF_{max} = 0.5, SF_{min} = 0.5\}. We limit the maximum fitness evaluation Max_{f_eval} = 2500.

As for RQ4, we have adopted the population size N = 20 with the maximum iteration Max_{iter} = 100 to ensure the same setting as the original benchmark experiments. In this case, the maximum fitness evaluation Max_{f_eval} = 2000.

Apart from the population size and maximum iteration, other algorithm-specific parameter settings of all constituent meta-heuristic algorithms are summarized in Table 1.

For statistical significance, we have executed HHGSO and all its constituent 30 times and reported the best and best–worst results as well as the best average using these...
[1. begin
[2. define algorithm list = {"Jaya", "SootyTern", "Owl", "Butterfly", "HenryGas"}
[3. define cluster N_size (default = 5)
[4. initialize all algorithms' defined parameters accordingly
[5. initialize population $X_1(i = 1, 2, \ldots, Max_{pop})$
[6. while (stopping criteria not met (i.e. $t < Max_{iteration}$))
[7. set division_idx=1;
[8. for each search agent (i.e. $i = 1, 2, \ldots, Max_{pop}$) do
[9. if (the first agent)
[10. current_running_algorithm=algorithm_list[division_idx]
[11. else if (i % N_size==0)
[12. division_idx++
[13. current_running_algorithm=algorithm_list[division_idx]
[14. end if
[15. if (current_running_algorithm ="Jaya")
[16. generate new candidate solution using Jaya based on Eq. (14)
[17. maintain cluster best $X_{cluster\ best}$
[18. else if (current_running_algorithm="SootyTern")
[19. update Sooty Tern parameters based on Eq. (15) until Eq. (23)
[20. generate new candidate solution using Sooty Tern based on Eq. (24)
[21. maintain cluster best $X_{cluster\ best}$
[22. else if (current_running_algorithm="Butterfly")
[23. update Butterfly parameters based on Eq. (25)
[24. generate new candidate solution using Butterfly based on Eq. (26) or Eq. (27)
[25. maintain cluster best $X_{cluster\ best}$
[26. else if (current_running_algorithm="Owl")
[27. update Owl parameters based on Eq. (28) until Eq. (30)
[28. generate new candidate solution using Owl based on Eq. (31)
[29. maintain cluster best $X_{cluster\ best}$
[30. else if (current_running_algorithm="HenryGas")
[31. update Henry Gas parameters based on Eq. (7) until Eq. (10)
[32. generate new candidate solution using Henry Gas based on Eq. (6)
[33. maintain cluster best $X_{cluster\ best}$
[34. end if
[35. end for
[36. select $N_w=$worst population based on Eq. (11) and update them using Eq. (12)
[37. select one $X_{best}$ from all $X_{cluster\ best}$
[38. if (non-improving cooperation $X_{global\ best\ new}$ is not better than $X_{global\ best\ old}$)
[39. update adaptive switching factor (SF) based on Eq. (13)
[40. if (random [0,1] <SF)
[41. resuffle ordering of algorithm list
[42. all left over clusters assigned to HenryGas (i.e. N_size>length of algorithm list)
[43. end if
[44. end if
[45. $t=t+1$
[46. break while loop when (fitness evaluation>=max_fitness_evaluation i.e. Max_{fit\ eval})
[47. end while
[48. return the global best solution $X_{best}$
[49. end

Fig. 5 Hybrid HGSO pseudocode
runs (as bold cells). Whenever possible, we also report the best average execution time (also as bold cells). For RQ1, we also reported the number of team members.

6.2 Case study objects selection and experimental procedure

Our case study objects relate to two Search-based Software Engineering problems, namely the team formation problem and the combinatorial t-way test suite generation. The discussion on RQ1 till RQ3 will be based on the experiments’ results, while the discussion on RQ4 and RQ5 will be based on the lessons learned from undertaking the work.

6.2.1 Team formation problem

The team formation problem can be seen as a set covering problem (SCP). Considered the NP-hard problem, the mathematical formulation of the set covering problem is as follows.

Let a universe of elements $E = \{e_1, \ldots, e_m\}$ and let the collection of subset $S = \{s_1, \ldots, s_n\}$ where $s_j \subseteq E$ and $\bigcup s_j = E$. Each set $s_j$ covers at least one element of $E$ and has an associated cost $c_j > 0$. The objective is to find a subcollection of sets $X \subseteq E$ that covers all of the elements in $E$ at a minimal cost.

Let $A^{m \times n}$ be a zero–one matrix where $a_{ij} = 1$ if element $i$ is covered by set $j$ and $a_{ij} = 0$ otherwise. Let $X = \{x_1, \ldots, x_n\}$ where $x_j = 1$ if set $s_j$ (with cost $c_j > 0$) is part of the solution and $x_j = 0$ otherwise.

Minimize $\sum_{j=1}^{n} c_j x_j$  \hspace{1cm} (32)

Subject to

$1 \leq \sum_{j=1}^{n} a_{ij} x_j, \quad i = 1, \ldots, m$ \hspace{1cm} (33)

$x_j \in \{0, 1\}$ \hspace{1cm} (34)

In the team formation problem, the goal is to form a team that covers all the required skills from the given search space of individual experts with certain defined skills. Based on the model from Lappas et al. [70], the costs of interaction between two experts ($A$ and $B$) can be calculated using Eq. (35).

Interaction Cost between Expert’s $A$ and $B$

$$= 1 - \frac{\text{Skills of } A \cap \text{Skills of } B}{\text{Skills of } A \cup \text{Skills of } B}$$ \hspace{1cm} (35)

The best team is the one with the most minimum interaction costs between experts in the team.

Table 1 Parameter settings values for algorithms

| Algorithms                                      | Parameters                                      | Value  |
|------------------------------------------------|------------------------------------------------|--------|
| Jaya Algorithm (JA)                            | No algorithm-specific parameters                | ~      |
| Sooty Tern Optimization Algorithm (STOA)       | Controlling variable ($C_f$)                    | [2, 0] |
|                                                | Random variable ($C_b$)                         | [0, 0.5]|
|                                                | Constants $u$ and ($v$)                         | 1      |
|                                                | Variable ($k$)                                 | [0, 2$\pi$]|
|                                                | Sensory modality ($c$)                          | [0, 1] |
|                                                | Power exponent ($a$)                            | [0, 1] |
|                                                | Switch probability ($p$)                        | 0.8    |
|                                                | Probability of vole movement ($p_{vm}$)         | [0, 1] |
|                                                | Uniformly distributed random number ($x$)       | [0, 0.5]|
|                                                | Linearly decreasing constant ($\beta$)          | [1.9, 0]|
|                                                | Cluster                                       | 5      |
|                                                | Constant ($l_1$)                               | 5$E$–02|
|                                                | Constant ($l_2$)                               | 100    |
|                                                | Constant ($l_3$)                               | 1$E$–02|
|                                                | Constant ($T^0$)                               | 298.15 |
|                                                | Constant ($K$)                                 | 1.0    |
|                                                | Influence of gas ($x$)                         | 1.0    |
|                                                | Constant ($\beta$)                             | 1.0    |
|                                                | Constant ($C_1$)                               | 0.1    |
|                                                | Constant ($C_2$)                               | 0.2    |
For RQ1 and RQ2, we subject HHGSO to two benchmark team formation problem data set involving IMDB [71] and DBLP [72]. The IMDB data set is a database of the movie actors and their roles by genre owned by Amazon. The cleaned data set consists of 1014 names of actors and unique 28 roles by genre. Meanwhile, the DBLP data set is a bibliographic database of scientific publications. The cleaned data set includes 5641 authors’ information with 3887 unique skills.

For our evaluation, we have adopted three sets of skills to look for. For IMDB, we adopt the 8, 16, and 24 skills set. Meanwhile, for DBLP, we have adopted the 30, 60, and 90 skills set. To ensure fairness of comparison, apart from running on the same platform, we have implemented the JA, STOA, BOA, OA, and HGSO using the Java programming language with the same data structure as HHGSO. For this reason, we are also able to report the time performance in addition to costs.

As far as RQ3 is concerned, there are two parts experiments. The first part of the experiment relates to the team formation problem. In this case, we subject HHGSO to 90 required skills from the faculty data set [73] (with 87 members and 141 defined skills) in order to find the best minimum switch factor (SFmin) and maximum switch factor (SFmax) with varying population size (i.e., parameters), and K is the number of possible values for the discrete variables.

Concerning the notation, the t-way test suite generation is often expressed in terms of Covering Array (CA) notation. The notation CA has four main parameters, namely S, t, p, and v (i.e., CA(S, t, v^p)). CA is a matrix of size S×P. Here, the symbols t refers to the interaction strength, S represents the test cases (rows), P is known as the number of parameters (columns) and v refers to the number of CA values for a specific P. For example, CA(S,2,3^4) can be seen as S×4 array that covers the test suite. In this case, the test suite covers t=2 with three v values and four p parameters, S=3×3=9 test cases.

For RQ3, we have subjected HHGSO to the following benchmark experiments (from Ref. [30]) as follows:

- A set of CA1(N;2,3^3), CA2(N;2,10^10), CA3(N;3,3^5), CA4(N;3,6^5), CA5(N;3,10^5); CA6(N;3,5^4⋅2⋅3^5)
- CA(N;2,3^4) where k is varied from 3 to 12
- CA(N;3,3^4) where k is varied from 4 to 12
- CA(N;4,3^4) where k is varied from 5 to 12
- CA(N;2,v^3) where v is varied from 2 to 7
- CA(N;3,v^3) where v is varied from 2 to 7
- CA(N;4,v^3) where v is varied from 2 to 7

The size performances for cooperative MASP-LLH algorithms (i.e., Exponential Monte Carlo hyper-heuristic with Counter, Modified Choice Function hyper-heuristic, Improvement Selection Rules hyper-heuristic, and Fuzzy Inference Selection hyper-heuristic) are taken directly from the original Ref. [30]. As no time performances are reported in the original reference, we also do not report our time performance for HHGSO. Unlike time performance (which depends on implementation language, data structure, system configuration, and the running environment), size performance is absolute. For this reason, the size performance can also give a meaningful indication of HHGSO performance against its cooperative MASP-LLH counterparts.

To address the second part of the experiment for RQ3, we have adopted the covering array of CA(N;t,5^2) for t=2 till 4 (from Ref. [74]). Here, we intend to find the best minimum switch factor (SFmin) and maximum switch factor (SFmax) for HHGSO with varying population size (i.e., cluster size = 4 and Maxfit eval = 2500) against the t-way test suite generation problem.

\[ \text{Minimize } f(Z) = \left| \{ I \in VIL : Z \text{ covers } I \} \right| \]

Subject to

\[ Z = Z_1, Z_2, \ldots, Z_i \in P_1, \ P_2, \ldots, P_i; \ i = 1, 2, \ldots, N \]

(36)
7 Results

Our results will be aligned to the given research questions as follows:

- RQ1: How is the performance of HHGSO compared to that of the original HGSO and its participating constituent algorithms?

Referring to Tables 2 and 3, HHGSO outperforms HGSO and all its constituent algorithms. To be specific, HHGSO obtains the best costs for all cases with the exception of DBLP with 30 required skills (see Table 3). Here, HGSO has the best costs (i.e., 1464.35), although having the same number of team members (i.e., 55) with HHGSO. In terms of average costs, HHGSO outperforms all the other algorithms. Unlike the best costs, which can be influenced by chance, average costs show the consistent performance of HHGSO as compared to other algorithms.

Putting HHGSO aside, HGSO comes in as the runner up. The fact that HHGSO and HGSO perform better than its constituent algorithms can be attributed to the fact that there is potentially more diversity in the population of solutions. HHGSO, in particular, enjoys five different candidate update operators (with different displacements) from Jaya, Sooty Tern Algorithm, Butterfly Optimization Algorithm, Owl Search Algorithm, and HGSO. The probabilistic changes in the update operators (owing to dynamic meta-heuristic-to-cluster mapping) ensure that the search process can easily go out of local optima.

The performance of Jaya, Sooty Tern Algorithm, and Owl Search Algorithm is at par with each other as they have mixed results as far as the average costs are concerned. Butterfly Optimization Algorithm performs the worst as its average costs are no better than any of the compared algorithms. On a positive note, the Butterfly Optimization Algorithm manages to outperform other algorithms in terms of the average execution time for the case of DBLP with 90 required skills. Overall, Owl Search Algorithm has the best average execution time.

- RQ2: What is the effect of cluster size with the meta-heuristic-algorithm-to-cluster mapping?

To answer RQ2, there is a need to deliberate on two main issues. The first issue relates to the effect of cluster size on the HHGSO overall performance of the search process. The second issue relates to the effect of cluster size on the algorithm mapping (e.g., whether or not there is a certain preference on a particular constituent algorithm).

Concerning the first issue, the metrics measurements for cluster size = 1 until 6 from Table 4 for IMDB and Table 5 for DBLP provides some indication on the effect of cluster size on the HHGSO overall performance. Specifically, we are interested in the performance of HHGSO with cluster size = 5. The reason is that there is a one-to-one mapping of each participating meta-heuristic algorithm with the defined cluster.

The given results show that HHGSO with cluster size = 5 outperforms HHGSO with other cluster sizes as far as the average cost is concerned. There is only one instance where HHGSO does not have the best average costs, that is, involving IMDB with 8 required skills (see Table 3). In this case, HHGSO with cluster size = 4 has

| Metrics measurement | JA | STOA | BOA | OSA | HGSO (cluster size = 5) | HHGSO (cluster size = 5) |
|---------------------|----|------|-----|-----|------------------------|------------------------|
| 8 required skills   |    |      |     |     |                        |                        |
| Best cost           | 4.32 | 4.04 | 4.26 | 4.37 | 3.81                   | 2.49                   |
| No of team members  | 4   | 4    | 4   | 4   | 3                      |                        |
| Ave cost            | 7.08 | 6.79 | 7.29 | 6.77 | 6.60                   | 6.58                   |
| Ave time (s)        | 3.25 | 3.05 | 3.20 | 3.03 | 3.00                   | 3.01                   |
| 16 required skills  |    |      |     |     |                        |                        |
| Best cost           | 13.98 | 13.89 | 20.65 | 15.97 | 12.96                 | 11.60                 |
| No of team members  | 7   | 7    | 8   | 7   | 6                      | 6                      |
| Ave cost            | 28.67 | 27.87 | 33.59 | 28.26 | 27.28                 | 26.73                 |
| Ave time (s)        | 3.20 | 3.30 | 3.24 | 3.25 | 2.99                   | 3.03                   |
| 24 required skills  |    |      |     |     |                        |                        |
| Best cost           | 17.71 | 24.22 | 32.11 | 28.40 | 23.76                 | 22.80                 |
| No of team members  | 7   | 8    | 9   | 9   | 8                      | 8                      |
| Ave cost            | 46.45 | 44.69 | 54.93 | 47.50 | 43.79                 | 43.61                 |
| Ave time (s)        | 3.58 | 3.26 | 3.34 | 3.31 | 3.05                   | 2.96                   |
outperformed HHGSO with cluster size = 5. We consider this as an outlier as HHGSO with cluster size = 4 is not performing well in other instances involving other given IMDB or DBLP datasets with other defined skills to find. The same observation can be seen as far as the best cost is concerned. HHGSO with cluster size = 5 gives the best cost for all cases with the exception of one case involving DBLP with 60 required skills (see Table 4). Again, the fact that HHGSO with cluster = 3 gives the best costs can also due to outlier as it does not perform well in other instances.

We conclude that as far as HHGSO with cluster size = 1 till 4 and 6 is concerned, there is no evidence of better performance than that of HHGSO with cluster size = 5 apart from having better execution times. When the cluster size definition is less than the participating algorithms, the algorithm-to-cluster mapping becomes too randomized, resulting in some algorithms be selected multiple times even though they may not be the performing ones (as a side effect of the dynamic switching factor). For this reason, the performance of HHGSO with cluster size < 5 is often poorer as compared to HHGSO with cluster size = 5. Having HHGSO variants with cluster size > participating algorithms also appears counter-productive as the extra cluster will be biased toward HHGSO update operators. This is reflected by the results tabulated in Tables 4 and 5, respectively.

It is interesting to note that in many parts of the results, the number of team members is the same for almost all cluster sizes. For example, in the case of IMDB with 8 required skills, HHGSO with cluster size-4, 5 and 6 has the same number of team members of 3 yet with different costs. The same observation can be seen for DBLP cases also. As the skills are not unique, a different combination of team is possible although at different costs.

Concerning the second issue, the average execution distribution for each participating constituent algorithms for cluster size = 1 until 6 are referred to in Table 4 for IMDB and in Table 5 for DBLP. Conveniently, the average execution distributions are plotted as cascaded bar charts in Figs. 6a–c and 7a–c, respectively. Firstly, from Figs. 6 and 7, all algorithms do have a chance to participate in the search process.

Considering HHGSO with cluster size = 1 till 3, no clear pattern of preferences exists for any particular meta-heuristic algorithm. This is expected as there are randomized many-to-one-assignment of the meta-heuristic algorithm to cluster(s). The pattern seems to change in the case of HHGSO with cluster size = 4, the distribution of each algorithm is nearly even because the mapping is now many-to-many (i.e., algorithms to clusters).

It is obvious that HHGSO with cluster size = 5 is expected to have 20% execution distribution each. This observation does not exactly materialize in the average distribution results (see Tables 4, 5). The main reason is that search execution may stop much earlier than maximum iteration, that is, when maximum fitness evaluation is reached. In fact, execution may stop in between any cluster execution (i.e., based on the maximum fitness evaluation). Therefore, some algorithms may have less execution than others (resulting in slightly less average execution).

Another obvious observation relates to the preference to HHGSO when the cluster size = 6. Here, there is one extra

| Metrics measurement | JA   | STOA | BOA  | OSA  | HGSO (cluster size = 5) | HHGSO (cluster size = 5) |
|---------------------|------|------|------|------|-------------------------|--------------------------|
| 30 required skills  |      |      |      |      |                         |                          |
| Best cost           | 342.64 | 319.63 | 368.68 | 343.43 | 318.93                 | **317.16**               |
| Ave cost            | 27    | **26** | 28   | 27   | 26                      | **26**                   |
| Ave time (s)        | 360.28 | 345.34 | 369.08 | 352.09 | 352.39                 | **340.10**               |
| 60 required skills  |      |      |      |      |                         |                          |
| Best cost           | 1464.86 | 1515.32 | 1522.00 | 1516.40 | **1464.35**          | 1469.29                  |
| Ave cost            | 55    | 56   | 56   | 56   | **55**                  | **55**                   |
| Ave time (s)        | 1537.14 | 1537.39 | 1574.25 | 1535.73 | 1520.11               | **1502.70**              |
| 90 required skills  |      |      |      |      |                         |                          |
| Best cost           | 3273.96 | 3195.21 | 3524.01 | 3268.25 | 3266.92               | **3035.01**              |
| Ave cost            | 82    | 81   | 85   | 82   | 82                      | **79**                   |
| Ave time (s)        | 3411.86 | 3330.80 | 3578.53 | 3406.82 | 3404.61               | **3327.03**              |
cluster more than the defined 5 participating constituent algorithms. This extra cluster by default is assigned to HGSO guaranteeing 2 clusters mapping out of a total 6 clusters, whereas other algorithms share only 1 mapping out of 6 clusters. As more and more clusters are defined and with fixed numbers of participating constituent algorithms, it is expected that the cluster mapping will be more and more biased toward HGSO.

- **RQ3**: What is the suitable minimum switch factor ($SF_{\text{min}}$) and maximum switch factor ($SF_{\text{max}}$) for HHGSO with varying population size?

For RQ3, we are interested on the range of $SF_{\text{min}}$ and $SF_{\text{max}}$ that give the best performances. Using the selected ranges of $SF_{\text{min}}$ and $SF_{\text{max}}$ (i.e., $\{SF_{\text{min}} = 0.1, SF_{\text{max}} = 0.9\}$, $\{SF_{\text{min}} = 0.2, SF_{\text{max}} = 0.8\}$, $\{SF_{\text{min}} = 0.3, SF_{\text{max}} = 0.7\}$, $\{SF_{\text{min}} = 0.4, SF_{\text{max}} = 0.6\}$ and $\{SF_{\text{min}} = 0.5,$
Tables 6 and 7 highlight the performances of HHGSO for the faculty data set involving 30, 60, and 90 required skills and $t$-way data set involving the $CA (N; t, 5^7)$ for $t = 2$ till 4. In Tables 6 and 7, the underline cells depict the best performance either in terms of the best value or the average value (all in bold format). Unlike the best value which can be influenced by luck owing to the randomness of the agent updates, the average value is often driven by consistency (i.e., low average means better overall performance). For this reason, the average value is more representative of the performance as compared to the best value.

Referring to the underline cells in both Tables 6 and 7, we can observe that the best performance is mostly achieved with the combination of $\{SF_{\text{min}} = 0.1, SF_{\text{max}} = 0.9\}$, and $\{SF_{\text{min}} = 0.2, SF_{\text{max}} = 0.8\}$, respectively.
Fig. 6 Average execution distribution based on IMDB data set for 8, 16, and 24 required skills

(a) Average execution distribution for 8 required skills
(b) Average execution distribution for 16 required skills
(c) Average execution distribution for 24 required skills

Fig. 7 Average execution distribution based on DBLP data set for 30, 60, and 90 required skills

(a) Average execution distribution for 30 required skills
(b) Average execution distribution for 60 required skills
(c) Average execution distribution for 90 required skills

Springer
Table 6 Cost performance of faculty data set for 30, 60, and 90 required skills with selected SF_max, SF_min and varying population size

| Cluster size = 4 | Maxfit eval = 2500 | SF_max = 0.9 | SF_max = 0.8 | SF_max = 0.7 | SF_max = 0.6 | SF_max = 0.5 |
|------------------|-------------------|--------------|--------------|--------------|--------------|--------------|
| Population size  | Metrics measurement |
| 30               | 30 required skills | 30           | 30           | 30           | 30           | 30           |
|                  | Best cost         | 228.15       | 228.15       | 228.15       | 228.15       | 228.15       |
|                  | No of team members| 22           | 22           | 22           | 22           | 22           |
|                  | Ave cost          | 228.16       | 228.16       | 228.17       | 228.19       | 228.18       |
|                  | 60 required skills | 60           | 60           | 60           | 60           | 60           |
|                  | Best cost         | 228.15       | 228.15       | 228.15       | 228.15       | 228.15       |
|                  | No of team members| 22           | 22           | 22           | 22           | 22           |
|                  | Ave cost          | 228.15       | 228.15       | 228.17       | 228.19       | 228.18       |
|                  | 90 required skills | 90           | 90           | 90           | 90           | 90           |
|                  | Best cost         | 228.15       | 228.15       | 228.15       | 228.15       | 228.15       |
|                  | No of team members| 22           | 22           | 22           | 22           | 22           |
|                  | Ave cost          | 228.15       | 228.15       | 228.17       | 228.19       | 228.18       |
There are only two instances when the performance is also commendable with \(SF_{\text{min}} = 0.3, SF_{\text{max}} = 0.7\) for the case of faculty data set with 90 required skills and CA \((N;3,5^7)\). No instances of best performance can be seen for \(SF_{\text{min}} = 0.4, SF_{\text{max}} = 0.6\), and \(SF_{\text{min}} = 0.5, SF_{\text{max}} = 0.5\).

Summing up, we can conclude that the best setting is \(SF_{\text{min}} = 0.2, SF_{\text{max}} = 0.8\) with population size = 30. This setting has consistently outperformed (or matched with the best of) other settings in terms of average performances (i.e., two instances in Table 6 and two instances in Table 7).

- **RQ4**: How good is the performance of HHGSO against its cooperative MASP-HLH counterparts (i.e., hyper-heuristic algorithms)?

Results from Tables 8, 9, 10, 11, 12, 13 and 14 indicate that the performance of HHGSO is at par with the Fuzzy Inference Selection hyper-heuristic algorithm (as another form of MASP-HLH derived from the hyper-heuristic family of algorithms). Reporting of many of new best in the literature, outperforming Fuzzy Inference Selection hyper-heuristic algorithm is a challenge. Despite such a challenge, HHGSO manages to get new overall best for CA \((N;2,10^3)\) and CA \((N;3,10^5)\) (in Table 6) as well as CA \((N;4,5^6)\) (in Table 14). The rest of the cells, more often than not, HHGSO only manage to equal the best results from the Fuzzy Inference Selection hyper-heuristic algorithm. On the positive note, HHGSO gives a better average test size than FIS in cells throughout Table 8 till Table 14, indicating its consistent performance. Putting Fuzzy Inference Selection hyper-heuristic aside, HHGSO outperforms all other compared hyper-heuristic algorithms.

**Table 6** (continued)

| Population size | Metrics measurement | 90 required skills |
|-----------------|---------------------|-------------------|
| 100             | Best cost           | 891.46 889.55     | 930.06 890.17 890.16 |
|                 | No of team members  | 43 43           | 44 43 43 |
|                 | Ave cost            | 936.44 935.10    | 934.32 953.64 949.78 |

| Cluster size = 4 | \(SF_{\text{min}} = 0.1\) | \(SF_{\text{max}} = 0.9\) | \(SF_{\text{min}} = 0.2\) | \(SF_{\text{max}} = 0.8\) | \(SF_{\text{min}} = 0.3\) | \(SF_{\text{max}} = 0.7\) | \(SF_{\text{min}} = 0.4\) | \(SF_{\text{max}} = 0.6\) | \(SF_{\text{min}} = 0.5\) | \(SF_{\text{max}} = 0.5\) |
| Population size  | Best Ave Best Ave Best Ave Best Ave Best Ave Best Ave Best Ave Best Ave Best Ave |
| 15              | 35 35.75 35 35.70 34 35.75 34 35.70 34 35.70 34 36.05 |
| 30              | 35 35.65 33 \underline{35.60} 35 35.60 35 35.60 35 35.60 35 35.60 |
| 50              | 35 \underline{35.60} 35 35.70 34 35.80 34 35.70 34 35.90 34 35.90 |
| 75              | 35 35.70 34 35.80 34 35.70 34 35.70 34 35.90 34 35.90 |
| 100             | 34 35.75 34 35.90 34 35.85 34 35.85 34 35.85 34 35.85 |

| Population size | Best Ave Best Ave Best Ave Best Ave Best Ave Best Ave Best Ave Best Ave Best Ave |
| 15              | 218 220.10 219 221.00 219 221.20 219 221.20 219 221.70 219 221.70 |
| 30              | 218 220.02 217 \underline{218.80} 218 221.20 218 220.60 218 220.70 |
| 50              | 217 218.90 217 219.05 217 219.60 218 220.60 219 220.50 219 220.50 |
| 75              | 217 219.30 \underline{215} 219.10 217 219.60 218 220.50 218 220.60 218 220.60 |
| 100             | \underline{215} 219.00 217 219.20 \underline{215} 219.02 219 220.80 218 220.20 218 220.20 |

| Population size | Best Ave Best Ave Best Ave Best Ave Best Ave Best Ave Best Ave Best Ave Best Ave |
| 15              | 1165 1172.20 1166 1169.40 1168 1173.10 1167 1172.90 1167 1172.40 |
| 30              | 1163 1167.80 1159 1163.80 1169 1175.50 1165 1170.05 1167 1171.10 |
| 50              | 1165 1165.60 1160 1164.30 1165 1176.20 1169 1173.30 1169 1173.80 |
| 75              | 1159 1163.20 1165 1165.90 1169 1173.10 1164 1169.60 1170 1173.05 |
| 100             | \underline{1155} \underline{1161.80} 1163 1168.05 1170 1174.05 1168 1170.70 1167 1172.10 |

[51x630]
Table 8 Size performance for selected CAs

| CA                     | Cooperative MASP-HLH | Exponential Monte Carlo with Counter | Modified choice function | Improvement selection rules | Fuzzy inference selection | Hybrid HGSO |
|------------------------|----------------------|-------------------------------------|--------------------------|-----------------------------|--------------------------|-------------|
|                        |                      | Best Ave                            | Best Ave                 | Best Ave                    | Best Ave                 | Best Ave    |
| CA_1 (N; 2, 3^13)      | 18                   | 19.05                               | 18                       | 19.45                       | 18                       | 18.65       | 17          | 18.80       |
| CA_2 (N; 2, 10^10)     | 155                  | 157.20                              | 157                      | 172.05                      | 156                      | 157.35      | 153         | 157.10      | 150         | 154.60      |
| CA_3 (N; 3, 3^6)       | 33                   | 38.85                               | 33                       | 38.90                       | 33                       | 37.75       | 33          | 38.20       | 33          | 38.00       |
| CA_4 (N; 3, 6^6)       | 323                  | 326.70                              | 323                      | 327.40                      | 322                      | 326.20      | 323         | 326.15      | 326         | 328.90      |
| CA_5 (N; 3, 10^6)      | 1485                 | 1496.50                             | 1483                     | 1499.25                     | 1482                     | 1486.80     | 1481        | 1486.20     | 1473        | 1478.90     |
| CA_6 (N; 3, 5^2/3^2)   | 100                  | 107.35                              | 100                      | 113.20                      | 100                      | 105.55      | 100         | 105.95      | 100         | 105.30      |

Table 9 Size performance for CA (N; 2, 3^k)

| k | Cooperative MASP-HLH | Exponential Monte Carlo with Counter | Modified choice function | Improvement selection rules | Fuzzy inference selection | Hybrid HGSO |
|---|----------------------|-------------------------------------|--------------------------|-----------------------------|--------------------------|-------------|
|   |                      | Best Ave                            | Best Ave                 | Best Ave                    | Best Ave                 | Best Ave    |
| 3 | 9                    | 9.83                                | 9                        | 9.70                        | 9                        | 9.67        | 9           | 9.81        |
| 4 | 9                    | 9.00                                | 9                        | 9.00                        | 9                        | 9.00        | 9           | 9.00        |
| 5 | 11                   | 11.24                               | 11                       | 11.30                       | 11                       | 11.23       | 11          | 11.23       |
| 6 | 14                   | 14.27                               | 13                       | 14.36                       | 13                       | 14.46       | 13          | 14.00       |
| 7 | 15                   | 15.07                               | 15                       | 15.23                       | 15                       | 15.10       | 14          | 15.07       | 15          | 15.10       |
| 8 | 15                   | 15.77                               | 15                       | 16.16                       | 15                       | 15.90       | 15          | 15.79       | 15          | 15.75       |
| 9 | 15                   | 16.23                               | 15                       | 16.43                       | 15                       | 16.10       | 15          | 15.97       | 15          | 16.05       |
| 10| 16                   | 17.10                               | 16                       | 17.20                       | 16                       | 17.50       | 16          | 17.03       | 16          | 17.01       |
| 11| 17                   | 18.90                               | 18                       | 18.50                       | 17                       | 18.30       | 16          | 17.45       | 16          | 17.60       |
| 12| 16                   | 17.96                               | 17                       | 18.29                       | 17                       | 18.40       | 16          | 17.80       | 16          | 17.79       |

Table 10 Size performance for CA (N; 3, 3^k)

| k | Cooperative MASP-HLH | Exponential Monte Carlo with Counter | Modified choice function | Improvement selection rules | Fuzzy inference selection | Hybrid HGSO |
|---|----------------------|-------------------------------------|--------------------------|-----------------------------|--------------------------|-------------|
|   |                      | Best Ave                            | Best Ave                 | Best Ave                    | Best Ave                 | Best Ave    |
| 4 | 27                   | 28.83                               | 27                       | 29.20                       | 27                       | 30.06       | 27          | 27.23       | 27          | 27.10       |
| 5 | 39                   | 41.47                               | 39                       | 41.40                       | 39                       | 41.60       | 37          | 41.30       | 39          | 41.55       |
| 6 | 33                   | 38.63                               | 40                       | 38.37                       | 33                       | 38.47       | 33          | 36.77       | 33          | 36.70       |
| 7 | 49                   | 50.46                               | 49                       | 50.50                       | 49                       | 50.47       | 48          | 50.40       | 48          | 50.40       |
| 8 | 52                   | 53.27                               | 52                       | 53.93                       | 52                       | 53.27       | 53          | 53.40       | 52          | 53.10       |
| 9 | 56                   | 57.79                               | 57                       | 58.07                       | 56                       | 57.87       | 56          | 57.77       | 56          | 57.60       |
| 10| 59                   | 61.17                               | 60                       | 60.77                       | 60                       | 60.10       | 59          | 61.03       | 59          | 61.00       |
| 11| 63                   | 63.87                               | 64                       | 65.27                       | 63                       | 63.67       | 63          | 63.53       | 63          | 63.53       |
| 12| 65                   | 67.61                               | 66                       | 68.13                       | 65                       | 66.93       | 65          | 66.13       | 65          | 66.20       |
function comes in joint third and closely followed by Exponential Monte Carlo with Counter.

- **RQ5:** Is there any overhead in terms time performance penalty of HHGSO implementation as compared to its constituent algorithms?

The time performance of HHGSO relates to its computational complexity metric. For this purpose, the Big O notation is used. Complexity is dependent on the number of search agents (\(n\)), the number of dimensions (\(d\)), the number of maximum iteration (Maxiter) and the fitness function evaluation (c).

As HHGSO consists of five meta-heuristic algorithms (i.e., HGSO, JA, STOA, BOA, OSA), its complexity \(O(HHGSO)\) is a combination of \(O(HGSO)\) + \(O(JA)\) + \(O(STOA)\) + \(O(BOA)\) + \(O(OSA)\). To analyze the overall time complexity, there is a need to analyze the contribution of each algorithm. With the cluster size = \(N\_size\), the time complexity of each algorithm is:

\[
O(HGSO) = O(\text{fitness function evaluation})
+ O(\text{agent update in memory}) + O(\text{dimension update})
= O(\text{Maxiter} \times c \times n/N\_size + \text{Maxiter} \times n/N\_size)
+ \text{Maxiter} \times d \times n/N\_size)
\]

\[
\geq O(\text{Maxiter} \times c \times n/N\_size + \text{Maxiter} \times d \times n/N\_size)
\]

(37)

\[
O(JA) = O(\text{fitness function evaluation})
+ O(\text{agent update in memory}) + O(\text{dimension update})
= O(\text{Maxiter} \times c \times n/N\_size + \text{Maxiter} \times n/N\_size)
+ \text{Maxiter} \times d \times n/N\_size)
\]

\[
\geq O(\text{Maxiter} \times c \times n/N\_size + \text{Maxiter} \times d \times n/N\_size)
\]

(38)

\[
O(STOA) = O(\text{fitness function evaluation})
+ O(\text{agent update in memory}) + O(\text{dimension update})
= O(\text{Maxiter} \times c \times n/N\_size)
+ \text{Maxiter} \times n/N\_size + \text{Maxiter} \times d \times n/N\_size)
\]

\[
\geq O(\text{Maxiter} \times c \times n/N\_size + \text{Maxiter} \times d \times n/N\_size)
\]

(39)

| Table 11 | Size performance for CA \((N; 4, 3^k)\) |
|---|---|
| Exponential Monte Carlo with Counter | Modified choice function | Improvement selection rules | Fuzzy inference selection | Hybrid HGSO |
| | Best | Ave | Best | Ave | Best | Ave | Best | Ave | Best | Ave |
| 5 | 81 | 84.23 | 81 | 89.07 | 81 | 88.27 | 81 | 87.27 | 81 | 87.11 |
| 6 | 130 | 133.33 | 129 | 133.83 | 129 | 134.17 | 129 | 134.10 | 130 | 133.10 |
| 7 | 149 | 154.27 | 151 | 155.17 | 147 | 153.53 | 147 | 153.90 | 147 | 153.67 |
| 8 | 172 | 174.96 | 173 | 175.47 | 171 | 174.83 | 171 | 174.47 | 171 | 174.41 |
| 9 | 160 | 187.87 | 142 | 190.53 | 171 | 190.33 | 159 | 189.47 | 178 | 189.05 |
| 10 | 206 | 209.00 | 205 | 208.83 | 206 | 208.77 | 206 | 208.67 | 205 | 208.22 |
| 11 | 221 | 224.67 | 222 | 226.13 | 221 | 224.33 | 221 | 223.13 | 221 | 223.13 |
| 12 | 237 | 238.51 | 237 | 239.21 | 236 | 238.11 | 235 | 237.43 | 235 | 237.60 |

| Table 12 | Size performance for CA \((N; 2, \nu^2)\) |
|---|---|
| Exponential Monte Carlo with Counter | Modified choice function | Improvement selection rules | Fuzzy inference selection | Hybrid HGSO |
| | Best | Ave | Best | Ave | Best | Ave | Best | Ave | Best | Ave |
| 2 | 7 | 7.00 | 7 | 7.00 | 7 | 7.00 | 7 | 7.00 | 7 | 7.00 |
| 3 | 15 | 15.13 | 15 | 15.13 | 15 | 15.17 | 14 | 15.00 | 14 | 15.00 |
| 4 | 24 | 25.07 | 24 | 25.47 | 23 | 25.00 | 24 | 24.87 | 24 | 24.40 |
| 5 | 34 | 35.83 | 34 | 36.63 | 34 | 35.90 | 34 | 35.70 | 34 | 35.40 |
| 6 | 48 | 49.00 | 48 | 49.67 | 47 | 49.51 | 47 | 48.75 | 47 | 48.40 |
| 7 | 64 | 65.93 | 64 | 66.85 | 64 | 66.25 | 64 | 65.65 | 64 | 65.45 |
Combining each algorithm’s contribution and generalizing with the number of participating meta-heuristic algorithms ($S$):

\[
O(BOA) = O(\text{fitness function evaluation}) + O(\text{agent update in memory}) + O(\text{new fragrance generation loop}) + O(\text{dimension update}) = O(\text{Maxiter} \times c \times n / N_{size} + \text{Maxiter} \times d \times n / N_{size})
\]

\[
O(OSA) = O(\text{fitness function evaluation}) + O(\text{agent update in memory}) + O(\text{dimension update}) = O(\text{Maxiter} \times c \times n / N_{size} + \text{Maxiter} \times d \times n / N_{size})
\]

\[
O(HHGSO) \cong S \times O(\text{Maxiter} \times c \times n / N_{size} + \text{Maxiter} \times d \times n / N_{size})
\]

Consider the limit when $N_{size} = 1$ and $S = 1$, then:

\[
O(HHGSO) \cong O(\text{Maxiter} \times c \times n + \text{Maxiter} \times d \times n)
\]

As can be seen from the aforementioned derivations [Eq. (37) till Eq. (43)], the general time complexity of HHGSO is similar to each of its constituents. The time complexity multiplier depends on the scaling factor = the number of participating algorithms.

- RQ6: How generalized can HHGSO implementation be for solving general optimization problems?

We have implemented and subjected HHGSO to two Search-based Software Engineering problems involving the team formation and the combinatorial $t$-way test suite generation. Our experimental results give a clear indication that the HHGSO approach can produce competitive results. Given that the two problems are domain-specific in nature,
the fact that HHGSO is able to optimize their problem-specific objective functions speaks volumes of its applicability to other optimization problems as well.

Generally, HHGSO tends to outperform general meta-heuristic algorithms. This could be due to the fact that HHGSO has the benefit of being able to utilize more than one meta-heuristic algorithm to perform the search process. Furthermore, the dynamic meta-heuristic-algorithm-to-cluster mapping usefully promotes exploration diversity whereby the displacement of any search agent in a cluster is not necessarily fixed to a particular type of update operator from one meta-heuristic algorithm only. In fact, such a feature could also be useful as a way to avoid entrapment in local optima.

8 Discussion

Reflecting on research questions given earlier, the usefulness of our approach can be debated further. Arguably, our implementation of HHGSO reveals three subtle properties. The first property relates to meta-heuristic-algorithm-to-cluster mapping. More specifically, meta-heuristic-algorithm-to-cluster mapping is dynamically mapped based on

![Fig. 8 Visual representation of HHGSO execution](image-url)
12 populations at any iteration = \( J \) 0cess. Figure 8 depicts the visual execution of HHGSO with cooperate in order to undertake the actual searching pro-

meta-heuristic algorithm not only to coexist but also to cooperate in order to undertake the actual searching process. Unlike the original HGSO, HHGSO allows more than one meta-heuristic algorithm not only to coexist but also to cooperate in order to undertake the actual searching process. Figure 8 depicts the visual execution of HHGSO with 12 populations at any iteration = \( J \) and \( K \). Here, it can be seen that the initial algorithm-to-cluster mapping varies with time based on the penalized and reward model.

The second property relates to the flexibility of HHGSO implementation as compared to its predecessor. Referring to Fig. 2 given earlier, HHGSO in the current form can be categorized as cooperative MASP-HLH (as all the algorithms work together as a unit in an unpredictable sequence and are non-intersecting to the host algorithm features). If the adaptive switching factor associated with the proposed cooperative penalized and reward model is removed from the implementation, the resulting HHGSO can be categorized as relay-based MASP-HLH. Taking the discussion further, consider the normal scenario when the cluster size is the same as the total list of participating meta-heuristic algorithms. Here, there will always be a unique one-to-one dynamic assignment of meta-heuristic-algorithm-to-cluster mapping at any iteration. In the case where the cluster size is less than the list of participating meta-heuristic algorithms, the dynamic assignment of meta-heuristic-algorithm-to-cluster mapping will be randomly decided among those algorithms. As such, at any particular iteration, there are algorithms that will not be participating in the meta-

heuristic-algorithm-to-cluster mapping (i.e., as cluster size < list of participating algorithms). Nonetheless, in the end, all the participating meta-heuristic algorithms do have a chance to run at least once during any of the iterations. This flexibility allows HHGSO to conveniently ride on many participating meta-heuristic algorithms without rigidly tied to a specific algorithm. For example, one can define 30 participating meta-heuristic algorithms with just 3 defined clusters. With the 3 defined clusters, there will be 3 defined meta-heuristic-algorithm-to-cluster mappings in each iteration. Here, the 3 defined meta-heuristic-algorithm-to-cluster mappings have the luxury to randomly adopt any 3 participating meta-heuristic algorithms based on penalized and reward model.

In one extreme case, even with only 1 defined cluster, we can still have cooperative MASP-HLH (i.e., considering many single algorithms in use although with just 1 cluster of the population). Similarly, at the other extreme, we can also have more clusters than the number of participating algorithms. In such a case, HHGSO will have more HGSO-to-cluster mappings than other meta-heuristic-algorithm-to-cluster mappings on the virtue of being the host algorithm. This aforementioned flexibility is unique to HHGSO implementation and is not found in any current hybridization scheme in the literature.

Finally, the third property relates to the simplicity and adaptability of the HHGSO approach. In the current work, HHGSO adopts a combination of meta-heuristic algorithms namely Jaya Algorithm (JA), Sooty Tern Optimization Algorithm (STOA), Butterfly Optimization Algorithm (BOA), and Owl Search Algorithm (OSA), respectively. The current implementation is straightforward in that HHGSO can seamlessly accommodate other meta-heuristic algorithms with minimal change in the source code implementation. This is a useful feature as the search space may vary depending on the optimization problems. For example, some optimization problems may have multiple peaks close to each other, thus, requiring more local search-based algorithms to avoid entrapment in local optima. With HHGSO, one can easily choose suitable algorithms as part of its constituents accordingly based on the need of the current problem at hand. In fact, any known meta-heuristic algorithms can be adopted as part of our HHGSO implementation.

9 Concluding remarks

In this paper, we have presented a new form of hybridization based on HGSO, termed HHGSO. In a nutshell, HHGSO divides the population into a set of clusters with dynamic meta-heuristic-algorithms-to-cluster mapping. To do so, HHGSO uses an adaptive switching factor based on a penalized and reward model to control the cluster mappings. The two features working together give HHGSO an edge over other approaches.

Based on our case study evaluations, we have concluded that HHGSO is sufficiently competitive and can be applied to other optimization problems. As the scope for future work, we are interested in applying HHGSO to other domain-specific optimization problems (e.g., traveling salesman, bin-packing, and vehicle routing problems) owing to its promising performances.

Acknowledgements The work reported in this paper is funded by the Malaysian Technical University Network (MTUN) Research Grant from the Ministry of Higher Education Malaysia titled: The Development of T-Way Test Generation Tool for Combinatorial Testing (Grant No: UIC19102). Bestoun S. Ahmed supported by the Knowledge Foundation of Sweden (KKS) through the Synergy Project AIDA - A Holistic AI-driven Networking and Processing Framework for Industrial IoT (Rek:20200067).
References

1. Vahidi B, Foroughi Nematolahi A (2019) Physical and physicochemical based optimization methods: a review. J Soft Comput Civ Eng 3(4):12–27. https://doi.org/10.22115/scce.2020.214959.1161
2. Lim SM, Leong KY (2018) A brief survey on intelligent swarm-based algorithms for solving optimization problems. In: Del Ser J, Osaba E (eds) Nature inspired methods for stochastic, robust and dynamic optimization, vol 47, pp 47–61. https://doi.org/10.5772/intechopen.76979
3. Lones MA (2020) Mitigating metaphors: a comprehensible guide to recent nature-inspired algorithms. SN Comput Sci 1(1):1–12. https://doi.org/10.1007/s42979-019-0050-8
4. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680. https://doi.org/10.1126/science.220.4598.671
5. Voudouris C (1998) Guided local search: an illustrative example in function optimisation. BT Technol J 16(3):46–50. https://doi.org/10.1023/A:100965513140
6. Pelot O (2017) A parallel variable neighborhood search for the vehicle routing problem with divisible deliveries and pickups. Comput Oper Res 85:71–86. https://doi.org/10.1016/j.cor.2017.03.009
7. Dueck G, Scheuer T (1990) Threshold accepting: a general purpose optimization algorithm appearing superior to simulated annealing. J Comput Phys 90(1):161–175. https://doi.org/10.1016/0021-9991(90)90201-B
8. Glover F, Laguna M (1998) Tabu search. In: Du D-Z, Pardalos PM (eds) Handbook of combinatorial optimization. Springer, Berlin, pp 2093–2229. https://doi.org/10.1007/978-1-4613-0303-9_33
9. Venkata Rao R (2016) Jaya: a simple and new optimization approach for global optimization. Soft Comput 20:165–181. https://doi.org/10.1007/s00500-015-1910-7
10. Dhiman G, Kaur A (2019) STOA: a bio-inspired based optimization algorithm for industrial engineering problems. Eng Appl Artif Intell 82:148–174. https://doi.org/10.1016/j.engappai.2019.03.021
11. Arora S, Singh S (2019) Butterfly optimization algorithm: a novel approach for global optimization. Soft Comput 23(3):715–734. https://doi.org/10.1007/s00500-018-3102-4
12. Jain M, Singh V, Rani A (2018) A novel nature-inspired algorithm for optimization: squirrel search algorithm. Swarm Evol Comput 44:148–175. https://doi.org/10.1016/j.swevo.2018.02.013
13. Hashim FA, Houssein EH, Mabrouk MS, Al-Atabany W, Mirjalili S (2019) Henry gas solubility optimization: a novel physics-based algorithm. Future Gener Comput Syst 101:646–667. https://doi.org/10.1016/j.future.2019.07.015
14. Zhao W, Zhang Z, Wang L (2020) Manta ray foraging optimization: an effective bio-inspired optimizer for engineering applications. Eng Appl Artif Intell 87:1–25. https://doi.org/10.1016/j.engappai.2019.103300
15. Kaveh A, Eslamlou AD (2020) Water strider algorithm: a new metaheuristic and applications. In: Proceedings of the structures, vol 25. Elsevier, Amsterdam, pp 520–541. https://doi.org/10.1016/j.istruc.2020.03.033
16. Yadav A, Kumar N (2020) Artificial electric field algorithm for engineering optimization problems. Expert Syst Appl 149:1–23. https://doi.org/10.1016/j.eswa.2020.113308
17. Faramarzi A, Heidarinejad M, Stephens B, Mirjalili S (2020) Equilibrium optimizer: a novel optimization algorithm. Knowl Based Syst 191:105190. https://doi.org/10.1016/j.knosys.2019.105190
18. Maciel O, Cuevas E, Navarro MA, Zaldívar D, Hinojosoa S (2020) Side-blotched lizard algorithm: a polymorphic population approach. Appl Soft Comput 88:1–24. https://doi.org/10.1016/j.asoc.2019.106039
19. Askari Q, Younas I, Saeed M (2020) Political optimizer: a novel socio-inspired meta-heuristic for global optimization. Knowl Based Syst 1–25, Art no. 105709. https://doi.org/10.1016/j.knosys.2020.105709
20. Moosavi SHS, Bardsiri VK (2019) Poor and rich optimization algorithm: a new human-based and multi populations algorithm. Eng Appl Artif Intell 85:165–181. https://doi.org/10.1016/j.engappai.2019.08.025
21. Kaur S, Awasthi LK, Sangal A, Dhiman G (2020) Tunicate swarm algorithm: a new bio-inspired based metaheuristic paradigm for global optimization. Eng Appl Artif Intell 90:1–29. https://doi.org/10.1016/j.engappai.2020.103541
22. Zhang Z, Ding S, Jia W (2019) A hybrid optimization algorithm based on cuckoo search and differential evolution for solving constrained engineering problems. Eng Appl Intell 85:254–268. https://doi.org/10.1016/j.engappai.2019.06.017
23. Zamli KZ, Din F, Ahmed BS, Bures M (2018) A hybrid Q-learning sine–cosine-based strategy for addressing the combinatorial test suite minimization problem. PLoS ONE 13(5):1–29. https://doi.org/10.1371/journal.pone.0195675
24. Zamli KZ (2018) Enhancing generality of meta-heuristic algorithms through adaptive selection and hybridization. In: Proceedings of the 2018 international conference on information and telecommunications technology. IEEE, pp 67–71. https://doi.org/10.1109/ictact.2018.8350825
25. Ahmed BS, Enoiu E, Afzal W, Zamli KZ (2020) An evaluation of Monte Carlo-based hyper-heuristic for interaction testing of industrial embedded software applications. Soft Comput 24(18):13929–13954. https://doi.org/10.1007/s00500-020-04769-z
26. Ayob M, Kendall G (2003) A Monte Carlo hyper-heuristic to optimise component placement sequencing for multi head placement machine. In: Proceedings of the international conference on intelligent technologies, InTech, vol 3, pp 132–141
27. Choong SS, Wong L-P, Lim CP (2019) An artificial bee colony algorithm with a modified choice function for the travelling salesman problem. Swarm Evol Comput 44:622–635. https://doi.org/10.1016/j.swevo.2018.08.004
28. Drake JH, Özcan E, Burke EK (2015) A modified choice function hyper-heuristic controlling unary and binary operators. In: Proceedings of the 2015 IEEE Congress on evolutionary computation (CEC), pp 3389–3396. https://doi.org/10.1109/cec.2015.7257315
29. Zamli KZ, Alkazemi BY, Kendall G (2016) A tabu search hyper-heuristic strategy for r-way test suite generation. Appl Soft Comput 44:57–74. https://doi.org/10.1016/j.asoc.2016.03.021
30. Zamli KZ, Din F, Kendall G, Ahmed BS (2017) An experimental study of hyper-heuristic selection and acceptance mechanism for combinatorial r-way test suite generation. Inf Sci 399:121–153. https://doi.org/10.1016/j.ins.2017.03.007
31. Jain M, Maurya S, Rani A, Singh V (2018) An owl search algorithm: a novel nature-inspired heuristic paradigm for global optimization. J Intell Fuzzy Syst 34:1573–1582. https://doi.org/10.3233/jifs-169452
32. Wolpert DH, Macready WG (1997) No free lunch theorems for search. IEEE Trans Evol Comput 1(1):67–82. https://doi.org/10.1109/4235.585893
33. Talbi E-G (2013) A unified taxonomy of hybrid metaheuristics with mathematical programming, constraint programming and machine learning. In: Hybrid metaheuristics. Springer, pp 3–76
34. Nasser AB, Zamil KZ, Alsewari AA, Ahmed BS (2018) Hybrid flower pollination algorithm strategies for t-way test suite generation. PLoS ONE 13(5):1–24. https://doi.org/10.1371/journal.pone.0195187. (in Eng)

35. Rambabu B, Reddy AV, Janakiraman S (2019) Hybrid artificial bee colony and monopoly butterfly optimization algorithm (HABC-MBOA)-based cluster head selection for WSNs. J King Saud Univ Comput Inf Sci. https://doi.org/10.1016/j.jsuci.2019.12.006

36. Long W, Cai S, Jiao J, Xu M, Wu T (2020) A new hybrid algorithm based on grey wolf optimizer and cuckoo search for parameter extraction of solar photovoltaic models. Energy Convers Manag 203:1–14. https://doi.org/10.1016/j.enconman.2019.112243

37. Yang X-S (2012) Flower pollination algorithm for global optimization. In: Proceedings of the international conference on unconventional computing and natural computation, vol 7445. Springer, Berlin, pp 240–249. https://doi.org/10.1007/978-3-642-32894-7_27

38. Karaboga D, Basturk B (2007) A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. J Glob Optim 39(3):459–471. https://doi.org/10.1007/s10898-007-9149-x

39. Wang G-G, Deb S, Cui Z (2019) Monarch butterfly optimization. Neural Comput Appl 31(7):1995–2014. https://doi.org/10.1007/s00521-015-2193-y

40. Mirjalili S, Mirjalili SM, Lewis A (2014) Grey wolf optimizer. Adv Eng Softw 69:46–61. https://doi.org/10.1016/j.advengsoft.2013.12.007

41. Yang X-S, Deb S (2009) Cuckoo search via lévy flights. In: Proceedings of the 2009 World Congress on nature and biologically inspired computing. IEEE, pp 210–214. https://doi.org/10.1109/nabc.2009.5393690

42. Alotaibi SS (2020) Optimization insisted watermarking model: hybrid firefly and jaya algorithm for video copyright protection. Soft Comput. https://doi.org/10.1007/s00500-020-04833-8

43. Sharma S, Saha AK, Ramasamy V, Sarkar JL, Panigrahi CR (2020) hBOSOS: an ensemble of butterfly optimization algorithm and symbiosis organisms search for global optimization. In: Advanced computing and intelligent engineering. Springer, pp 579–588

44. Mirjalili S (2016) SCA: a sine cosine algorithm for solving optimization problems. Knowl Based Syst 96:120–133. https://doi.org/10.1016/j.knosys.2015.12.022

45. Zames G et al (1981) Genetic algorithms in search, optimization and machine learning. Inf Technol J 3(1):301–302

46. Yang X-S (2008) Firefly algorithm. In: Nature-inspired metaheuristic algorithms, vol 20, pp 79–90

47. Cheng M-Y, Prayogo D (2014) Symbiotic organisms search: a new metaheuristic optimization algorithm. Comput Struct 139:98–112. https://doi.org/10.1016/j.compstruc.2014.03.007

48. Drake JH, Kheter A, Özcan E, Burke EK (2019) Recent advances in selection hyper-heuristics. Eur J Oper Res. https://doi.org/10.1016/j.ejor.2019.07.073

49. Ahmed BS, Enoiu E, Afzal W, Zamil KZ (2020) An evaluation of monte carlo-based hyper-heuristic for interaction testing of industrial embedded software applications. Soft Comput. https://doi.org/10.1007/s00500-020-04769-z

50. Zamil KZ, Din F, Baharom S, Ahmed BS (2017) Fuzzy adaptive teaching learning-based optimization strategy for the problem of generating mixed strength t-way test suites. Eng Appl Artif Intell 59:35–50. https://doi.org/10.1016/j.engappai.2016.12.014

51. Din F, Zamil KZ (2018) Hyper-heuristic based strategy for pairwise test case generation. Adv Sci Lett 24(10):7333–7338. https://doi.org/10.1166/asl.2018.12938

52. Luan J, Yao Z, Zhao F, Song X (2019) A novel method to solve supplier selection problem: hybrid algorithm of genetic algorithm and ant colony optimization. Math Comput Simul 156:294–309. https://doi.org/10.1016/j.matcom.2018.08.011

53. Noori S, Ghannadpour SF (2012) High-level relay hybrid metaheuristic method for multi-depot vehicle routing problem with time windows. J Math Model Algorithms 11(2):159–179. https://doi.org/10.1007/s10852-011-9171-3

54. Lepagnot J, Idoumghar L, Brévilliers M, Idrissi-Aouad M (2017) A new high-level relay hybrid metaheuristic for black-box optimization problems. In: Proceedings of the international conference on artificial evolution. Springer, Berlin, pp 115–128. https://doi.org/10.1007/978-3-319-78133-4_9

55. Dorigo M, Gambardella LM (1997) Ant colonies for the traveling salesman problem. Biosystems 43(2):73–81. https://doi.org/10.1016/S0303-2647(97)01708-5

56. Lepagnot J, Nakib A, Oulhadj H, Siarry P (2013) A multiple local search algorithm for continuous dynamic optimization. J Heuristics 19(1):35–76. https://doi.org/10.1007/s10732-013-9215-0

57. Viktorin A, Pluhacek M, Senkerik R (2016) Success-history based adaptive differential evolution algorithm with multi-chao- tic framework for parent selection performance on CEC2014 benchmark set. In: Proceedings of the 2016 IEEE Congress on evolutionary computation (IEEE CEC). IEEE, pp 4797–4803. https://doi.org/10.1109/cec.2016.7744404

58. Zambrano-Bigiarini M, Clerc M, Rojas R (2013) Standard particle swarm optimisation 2011 at CEC-2013: a baseline for future PSO improvements. In: Proceedings of the 2013 IEEE Congress on evolutionary computation. IEEE, pp 2337–2344. https://doi.org/10.1109/cec.2013.6557848

59. Rao RV, Savsani V, Balic J (2012) Teaching–learning–based optimization algorithm for unconstrained and constrained real-parameter optimization problems. Eng Optim 44(12):1447–1462. https://doi.org/10.1080/0305215X.2011.652103

60. Pourvaziri H, Naderi B (2014) A hybrid multi-population genetic algorithm for the dynamic facility layout problem. Appl Soft Comput 24:457–469. https://doi.org/10.1016/j.asoc.2014.06.051

61. Zhou J, Yao X (2017) Multi-population parallel self-adaptive differential artificial bee colony algorithm with application in large-scale service composition for cloud manufacturing. Appl Soft Comput 56:379–397. https://doi.org/10.1016/j.asoc.2017.03.017

62. Chen H, Heidari AA, Chen H, Wang M, Pan Z, Gandomi AH (2020) Multi-population differential evolution-assisted Harris Hawks optimization: framework and case studies. Future Gener Comput Syst. https://doi.org/10.1016/j.future.2020.04.008

63. Storn R, Price K (1997) Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J Global Optim 11(4):341–359. https://doi.org/10.1023/A:100820821328

64. Heidari AA, Mirjalili S, Farihasi H, Aljaraah I, Mafjarah M, Chen H (2019) Harris Hawks optimization: algorithm and applications. Future Gener Comput Syst 97:849–872. https://doi.org/10.1016/j.future.2019.02.028

65. Cruz-Chávez MA et al (2019) Hybrid micro genetic multi-population algorithm with collective communication for the job shop scheduling problem. IEEE Access 7:82338–82376. https://doi.org/10.1109/ACCESS.2019.2924218

66. Lapa K, CPalka K, Paszkowski J (2019) Hybrid multi-population based approach for controllers structure and parameters selection. In: Proceedings of the international conference on artificial intelligence and soft computing. Springer, Berlin, pp 456–468. https://doi.org/10.1007/978-3-303-00912-4_42

67. Brown TL (2009) Chemistry: the central science. Pearson Education, New Delhi
68. Mohebbi V, Naderifar A, Bebbahani R, Moshfeghian M (2012) Determination of Henry’s law constant of light hydrocarbon gases at low temperatures. J Chem Thermodyn 51:8–11. https://doi.org/10.1016/j.jct.2012.02.014

69. Staudinger J, Roberts PV (1996) A critical review of Henry’s law constants for environmental applications. Crit Rev Environ Sci Technol 26(3):205–297. https://doi.org/10.1080/10643389609388492

70. Lappas T, Liu K, Terzi E (2009) Finding a team of experts in social networks. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining, Paris, France, 2009. Association for Computing Machinery, pp 467–476. https://doi.org/10.1145/1557019.1557074

71. IMDB (2020) “IMDB Dataset”. https://github.com/MAK660/Dataset/blob/master/IMDB_DataSet.txt. Accessed 15 May 2020

72. T. D. Team (2019) “DBLP Dataset”. https://github.com/MAK660/Dataset/blob/master/DBLP_DataSet.txt. Accessed Nov 2019

73. UMP. “Faculty Staff Expertise FKOM data set.” https://github.com/MAK660/Dataset/blob/master/Staff_Expertise_DataSet.txt. Accessed 10 Jan 2020

74. Hassan AA, Abdullah S, Zamli KZ, Razali R (2020) Combinatorial test suites generation strategy utilizing the whale optimization algorithm. IEEE Access 8:192288–192303. https://doi.org/10.1109/ACCESS.2020.3032851

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.