Research Article

An adaptive structure on a new local branching algorithm using instantaneous dimensions and convergence speed: a case study for multi-commodity network design problems

Hooria Hajiyan[1] · **Masoud Yaghini**[2]

Received: 10 October 2019 / Accepted: 29 April 2020 / Published online: 9 May 2020

Abstract
This study presents an adaptive parameter control structure for the New Local Branching Algorithm (NLBA) to enhance the overall efficiency of the algorithm by implementing an exploitation and exploration approach within solution search subspaces. In other words, the optimization problem is divided into several sub-problems and the proposed adaptive parameter control structure introduces some deterministic rules to check the convergence speed and adapt the performance of the algorithm in each subspace. So, the proposed adaptive structure uses an exploration approach to avoid becoming trapped at local optimum that may not be the global optimum, as well as an exploitation approach to look near good solutions for even better ones at each iteration. To evaluate the performance of the proposed adaptive parameter control structure, 26 multi-commodity network design problems are tested. The experimental results of this adaptive structure for NLBA (ANLBA) confirm that its performance compared to the original NLBA, a hybrid structure that consists of the Self-Adaptive Harmony Search (SAHS) algorithm as a parameter tuning method for NLBA, and CPLEX is significantly improved.

Keywords Local branching algorithm · Parameter setting · Adaptive parameter control · Exploitation · Exploration

1 Introduction

Combinational optimization problems are known to encounter many challenges including discrete decision variables and a large load of binary variables. Indeed, in more involved, real-world applications, finding an exact solution can lead to high computational costs, so metaheuristic algorithms are often used instead. A metaheuristic algorithm has several key parameters that affect the performance of the algorithm. Accordingly, an appropriate parameter setting helps to enhance the process of searching the solution space and lower computational cost. Given the importance of such parameters, two general approaches to the parameter setting have been proposed, namely offline and online procedures. Firstly, online procedures, hereinafter referred to as parameter control methods, adjust parameter values based on real-time feedback received from the algorithm. Contrarily, the offline methods treat the parameters as fixed constants during the execution of the algorithm.

A review of the literature related to parameter setting for metaheuristic algorithms shows that an appropriate parameter setting provides large contributions to improved performance of such algorithms [1]. Focusing on the dependence of the algorithm performance on the parameter setting, offline methods have been generally classified into two classes depending on the use of explicit models: model-free and model-based [2]. On the other hand, the parameter control approach can be categorized based on two major aspects: the mechanism

✉ Hooria Hajiyan, hajiyanh@mcmaster.ca | [1]McMaster University Continuing Education, Hamilton, ON, Canada. [2]Department of Rail Transportation Engineering, Iran University of Science and Technology, Tehran, Iran.

by which parameters are changed, and the effects of that mechanism on the algorithm. Regarding the mechanisms by which parameters are changed, there are three different ways for changing the value of a parameter in the parameter control methods [3]. The first approach called deterministic parameter control which uses some rules to change the value of parameters without getting any feedback from the performance of the algorithm in the search space. The second approach is adaptive parameter control which takes some feedback during the execution of the algorithm and uses that to set a framework and determine the direction of the change in the trend of the algorithm for next iterations according to implicit or explicit rules. The last one is the self-adaptive parameter control approach which is founded on evolutionary algorithms, where the evolution of successive generations is used to implement a self-adaptive process for parameter setting based on the idea that better values of the parameters lead to better solutions [3, 4].

Following comprehensive research on the adaptive parameter control for evolutionary algorithms, Tvrdik [5] used a new conceptual model that subdivided the process of parameter adaptation into several steps. In addition to this, some researchers emphasized the role of the Differential Evolution Algorithm (DEA) to formulate adaptive and self-adaptive procedures [6–8], and it has been shown that the performance of DEA is significantly affected by the choice of mutation strategies and control parameters [9]. Contrarily, a completely different mechanism in a differential evolution algorithm was presented by Zamuda and Brest [6] who provided new insight into adaptation and self-adaptation by focusing on iteration-temporal randomness. That is when the mechanism generates new values for a control parameter, randomness of such value generation is controlled by a randomness level parameter which controls the dynamics of the parameter values and improves their propagation through suitable individuals. Despite the presence of numerous methods for parameter setting, the parameter control approach, especially with its adaptive and self-adaptive tools, provides high-accuracy changes for adaptation to the trend of the algorithm [5, 7, 8, 10–13].

Over the last decade of investigations on adaptive parameter control approaches in which they should dynamically adapt to ever-changing dimensions of the problem, Jansen et al. implemented parameter control based on offspring population size [14]. In this approach, the role of this parameter was determined and a simple way was proposed to dynamically change its value based on an adaptive procedure. Interestingly, the results confirmed that the application of variable offspring population size reduced the number of generations remarkably with no significant increase in the computational effort. Later, Lagos et al. [15] proposed a framework based on a genetic programming technique to find optimal values of parameters for a Tabu search algorithm. This adaptive approach was also implemented on a new swarm intelligence algorithm called Firefly Algorithm (FA) to formulate a modified version of FA called FA with adaptive control parameters (APFA) [16] and coupled FA with a fuzzy method to dynamically adapt algorithm parameters to the problem [17]. In another investigation on a well-known optimization algorithm called Particle Swarm Optimization (PSO), Wang et al. [10] proposed a robust and self-adaptive learning-based algorithm that could generate initial generations of solutions with much better quality. More recently, Deng et al. [18, 19] improved the nature-inspired optimization algorithm by a multi-population strategy, a pheromone updating strategy, and a pheromone diffusion mechanism to balance convergence speed as a parameter control method. In addition to this, Nseef et al. proposed an adaptive parameter control approach using fuzzy logic to dynamically adjust the parameter values concerning normalized iteration and an error value. To cope with such dynamic changes, [20] proposed an adaptive multi-population Artificial Bee Colony (ABC) algorithm. Another adaptive control approach for the ABC algorithm was proposed in Ref [11]. to enhance algorithm performance, ultimately formulating a self-adaptive ABC based on the global best candidate (SABC-GB). The literature also describes several adaptive versions of well-known metaheuristic algorithms for parameter setting. For instance, Xia et al. [21] proposed Dynamic Ant Colony Optimization (ACO) to improve converge speed and solution quality. Castillo et al. [22] implemented an adaptive parameter control based on a fuzzy approach to dynamically control a particular parameter in an ACO algorithm. Another endeavor to improve the quality of ACO was presented by Deng et al. [23] who introduced a genetic and ant colony adaptive collaborative optimization (MGACACO) algorithm for solving complex problems. This algorithm combined the exploration capability of Genetic Algorithm (GA) and the stochastic capability of ACO as an adaptive tool for parameter control to make a uniform pheromone distribution and find the optimal solution.

Exploration and exploitation are common in adaptive parameter control approaches to enhancing the convergence of an algorithm. For example, Liu et al. proposed a learning-based PSO by reinforcement learning to control its exploration and exploitation, and [24] introduced adaptive PSO with supervised learning and control (APSOSLC). Similarly [25, 26], improved PSO based on an exploring strategy which can be dependently adjusted. Furthermore, [27] Attempted to set parameters of PSO adaptively while maintaining the diversity to improve its exploration capability which is similar to research done by Song et al.

[28] which led to enhanced PSO by an exploration and exploitation approach to increase the algorithm's robustness against premature convergence. Also, [29] defined adaptive PSO with four evolutionary states: exploration, exploitation, convergence, and jumping out. Inspired by supervised learning and predictive control strategies in machine learning, this approach aimed at extending the swarm diversity and speeding up the convergence. On another note, [30] Improved moth-flame optimization algorithm by adopted Cauchy mutation to enhance the global exploration ability. Similarly, Xu et al. [31] proposed a new framework for the moth-flame optimization algorithm by Gaussian mutation coupled with chaotic local search to get a more stable balance between exploration and exploitation. Relatedly, Lin and Gen [4] focused on how to strike a balance between the exploration and exploitation of the solution space in different algorithms like free flower based on a pairwise testing strategy [32]. Due to the various dimension of different problems, the balance should be changed dynamically regarding the current status of the evolution process [33].

The main goal of this study is to propose an adaptive parameter control structure able to improve the search process while decreasing computational cost through changing parameters based on exploration and exploitation within solution subspaces. When it comes to the many binary variables in Mixed Integer Programming (MIP) applications, the adaptive parameter control approach can keep from becoming trapped at a local optimum by dynamically adjusting the algorithm through getting feedback. In the present paper, an adaptive parameter control structure is proposed for the New Local Branching Algorithm (NLBA) [34] which is an extension to the Local Branching Algorithm (LBA) [35]. In other words, we proposed an adaptive structure in which an optimization problem is divided into several sub-problems to solve. So, the proposed adaptive parameter control structure checks the status of the given feasible solution and the elapsed processing time at each iteration to decide on exploration or exploitation of the search subspace for the next iterations.

The paper is organized as follows. Firstly, Sect. 2, presents LBA and the NLBA together with the corresponding pseudocode. Then, in Sect. 3, the proposed adaptive parameter control structure for the NLBA, and different solutions that are likely to show up during the execution of the algorithm are thoroughly explained. Following this, Sect. 4 presents a discussion on the mathematical formulation of the multi-commodity network design problems. In Sect. 5, the performance of the proposed adaptive parameter control structure is evaluated and 26 experimental instances are tested. In this section, the proposed adaptive parameter control structure is compared to the original NLBA, the NLBA using Self-Adaptive Harmony Search (SAHS) to tune its parameters, and CPLEX. Finally, Sect. 6 presents a discussion of experimental results and conclusions, and references are listed in Sect. 7.

2 NLBA and the original LBA framework

First proposed by Fischetti and Lodi [35], LBA is a technique for solving MIP problems. Although it is known as an exact solution method, it becomes heuristic by redefining some control parameters to find high-quality solutions utilizing a MIP solver. In practice, this method uses the MIP solver as a black box tool to effectively explore the solution subspaces defined at a strategic level by the external branching framework as a Local Branching Cut (LBC).

To clarify the framework of LBA, a generic MIP model with 0–1 variables and the additional constraint LBC are shown in Eq. (1) to Eq. (5).

The general framework of LBA can be defined as follows [35]:

Equation (1) is the objective function (P) with the best feasible solution x which satisfies the constraint of Eq. (2). Consider j to be the index of a MIP variable. Concerning Eqs. (3) and (4), the variables are divided into the sets β and δ, where β contains binary variables, and δ contains general integer and continuous variables. If $\bar{x}$ is a given feasible solution of (P), for each possible incumbent solution $\bar{x}$ of (P) and a positive integer parameter k, the k-OPT neighborhood N ($\bar{x}$, k) of $\bar{x}$ would be the set of feasible solutions of (P) satisfying the additional constraint LBC, Eq. (5). The two terms in the left-hand side of the Eq. (5) count the number of binary variables flipping their values to $\bar{x}$, either from 1 to 0 or from 0 to 1, respectively.

$$\text{(P) Min } C^T x \tag{1}$$

$$\text{s.t } Ax \geq b \tag{2}$$

$$x_j \in \{0, 1\}\ \forall j \in \beta \neq \emptyset \tag{3}$$

$$x_j \neq 0\ \forall j \in \delta \tag{4}$$

$$\Delta\left(x, \bar{x}\right) = \sum_{j \in \beta : \bar{x}_j = 1} (1 - x_j) + \sum_{j \in \beta : \bar{x}_j = 0} x_j \leq k \tag{5}$$

The purpose of the LBA is that the LBC is a branching criterion within an enumerative scheme for (P). So, for the given incumbent solution $\bar{x}$, the solution space associated with the current branching node can be partitioned through a disjunction, Eq. (6).

$$\Delta(x,\bar{x}) \leq k \text{ (left branch) or } \Delta(x,\bar{x}) \geq k+1 \text{ (right branch)} \tag{6}$$

The value of parameter *k* should be chosen as the largest value producing a left branch sub-problem which is likely to be solved more easily than the one associated with its parent. The idea is that the neighborhood $N(\bar{x}, k)$ corresponding to the left branch must be small enough to be optimized within a short processing time, but still large enough to have a high probability of containing better solutions than $\bar{x}$. But only the right branches get away from the previously-seen solution subspace without calling the black box to solve the sub-problem.

As mentioned before, LBA is a great technique for MIP problems and has been used in numerous research works. For instance, Fallahia et al. [36] used it for solving the Set Covering Problems utilizing the Design of Experiments (DOE) as a parameter tuning method. In another work, Yaghini et al. [37] further improved LBA for train formation planning.

During recent years, a new hybrid algorithm was proposed as an extension to the LBA: New Local Branching Algorithm (NLBA) [34]. This algorithm also used the MIP solver as a black box, but the proposed external local branching framework made the solving process much easier. In other words, the right branch of the Eq. (6) was removed and the left one was taken as an LBC at each iteration. Finally, the structure of the NLBA and the respective pseudocode is shown in Fig. 1. The NLBA parameters are also described in Table 1.

In the pseudocode of Fig. 1, *TL*, *zeroIterationTime*, *increaseRateOfRHSForImprove, initialRHS*, and *increaseRateOfRHSForNotImprove* are initialized. Then, the algorithm starts with an initial solution that could be generated by different neighborhood structures. In the basic version of the LBA, the MIP solver begins to solve the problem at a certain zero iteration time, *zeroIterationTime*, and follows an iterative process. Once the new feasible solution is seen not to improve the previous one, node zero is terminated, the *zeroIterationSolution* ($\bar{x}$) is considered as the *bestSoFarSolution* ($\bar{x}$), the objective value of the reference solution $\bar{x}$ is considered as an initial solution, and the value of *bestSoFar* is updated. Then, the LBC is built based on the reference solution $\bar{x}$.

This method consists of the main while loop which is iterated until one of the termination criteria is met. More precisely, at each iteration, the MIP solver receives two parameters as inputs, namely the total processing time limit *TL*, and the *bestSoFar*, which is used as an upper bound *UB* to interrupt the optimization as soon as the best lower bound becomes greater than or equal to *UB*. Hence, at each iteration, the MIP solver is aborted as soon as the best *iterationObjective* is found by the given reference solution $\bar{x}$.

Fig. 1 Pseudocode for the NLBA

Pseudocode for NLBA

```
1   Initialize TL, zeroIterationTime, initialRHS, increaseRateOfRHSForImprove,
    increaseRateOfRHSForNotImprove;
2   Set iterationObjective = bestSoFar = +∞;
3   Create model;
4   Solve the iteration zero(zeroIterationTime);
5   Set bestSoFarSolution ($\bar{x}$)= zeroIterationSolution ($\bar{x}$);
6   Set currentRHS = initialRHS;
7   Add the constraint  $(x,\bar{x})\Delta\ll currentRHS$to the MIP model;
8   While (termination criteria not satisfied) do
9   Solve the model(TL, UB);
10      If (iterationObjective < bestSoFar)
          a.  Update bestSoFar;
          b.  Update reference solution ($\bar{x}$);
          c.  Update currentRHS with increaseRateOfRHSForImprove;
          d.  Delete the last constraint from the MIP model;
          e.  Add the constraint $\Delta(x,\bar{x}) \ll currentRHS$to the MIP model;
11      Else
          a.  Update currentRHS with increaseRateOfRHSForNotImprove;
          b.  Update the last constraint with increaseRateOfRHSForNotImprove;
12      Endif;
13      Endwhile;
```

Table 1 NLBA Parameters description

Parameter	Description
TL	Total considered time for solving each problem
zeroIterationTime	Initial time, the time at which iteration zero is executed
initialRHS	Initial right-hand side value of the LBC
currentRHS	Current value of the LBC
bestSoFarSolution $(\bar{x})$	Best feasible solution found (reference solution)
bestSoFar	The objective value of the best feasible solution
iterationObjective	Value of the given feasible solution at each iteration
increaseRateOfRHSForImprove	Rate of increase of the right-hand side value of the LBC when the *bestSoFar* is getting updated
increaseRateOfRHSForNotImprove	Rate of increase of the right-hand side value of the LBC when the *bestSoFar* is not getting updated

Next, if the *iterationObjective* can improve the *bestSoFar*, the value of *bestSoFar* is updated, the last LBC is deleted, the new constraint is rebuilt by the new reference solution $\bar{x}$, and the right-hand side value *currentRHS* is increased by the *increaseRateOfRHSForImprove*. On the other hand, if the *iterationObjective* fails to improve the *bestSoFar*, the *currentRHS* is just updated by the *increaseRateOfRHSForNotImprove*.

Regarding the performance of the NLBA, we found that the following three key points affect the process of searching the solution space and can help find better solutions:

- The right-hand side value of LBC likely exceeds the number of binary variables, in which case the algorithm is stopped before the total processing time limit is reached.
- Extension of the search subspaces in successive iterations makes the searching process more difficult while lowering the chance of finding the optimal solution. Therefore, it is a good idea to start exploitation around the best points and occasionally use the exploration method to find unforeseen new points along the whole search space.
- In a situation where the objective value has not improved, an extension of the search subspace without regarding the status of the given solution is not so efficient. Accordingly, it is better to consider the status of the given solution before widening or narrowing the search subspace.

Considering all of the aforementioned reasons, we introduce an adaptive parameter control structure based on exploitation and exploration for the NLBA in the following section.

3 The proposed adaptive parameter control structure for the new local branching algorithm

Given the major effects of the parameters of an algorithm on its performance, this section introduces an adaptive parameter control structure for the NLBA (ANLBA). For this purpose, beginning with the identification of the main parameters affecting the algorithm, an adaptive parameter control structure is proposed based on exploitation and exploration within the solution space. However, such exploitation and exploration is difficult because algorithm performance is highly problem dependent [4]. Thus, the parameter control procedure should occur dynamically. The parameters of ANLBA and the pseudocode of its adaptive procedure are shown in Table 2 and Fig. 2, respectively.

In this pseudocode, *TL*, *iterationTime*, *initialRHS*, *firstImproveRHS*, *increaseRateOfRHS*, *increaseRateOfTime*, *iterationStatus*, *bestSoFarSolution*$(\bar{x})$ and *bestSoFar* are total processing time limit, a certain iteration time, the initial right-hand side value, the right-hand side value of the LBC when the first improvement occurs than to the initial objective value, the rate of increase right-hand side value of the LBC, the rate of increase the iteration time limit, status of the given solution upon calling the MIP solver, reference solution $(\bar{x})$, and the objective value of the reference solution, respectively.

In the first part of the pseudocode, the variables are initialized and then the model is created. Since there are too many methods for generating an initial solution and given that this adaptive structure looks for the first improvement in the objective value upon executing the algorithm, the method used to generate the initial

Table 2 ANLBA Parameters description

Parameter	Description
TL	Total considered time for solving each problem
IterationTime	Considered time for solving each iteration
initialRHS	Initial right-hand side value of the LBC
firstImproveRHS	Right-hand side value of LBC when the first improvement relative to the initial objective occurs
currentRHS	Current value of the LBC
bestSoFarSolution ($\bar{x}$)	Best feasible solution found (reference solution)
bestSoFar	The objective value of the best given feasible solution
iterationObj	Value of the given feasible solution in each iteration
iterationStatus	Status of the given solution upon calling the MIP solver
increaseRateOfRHS	Rate of increase of the current LBC
increaseRateOfTime	Rate of increase of iteration time

solution and its value is of paramount importance. The reason for considering the first improvement after executing the NLBA is that the proposed adaptive structure can adjust its performance with the problem dimensions and try not to change many binary variables without producing any improvement. Therefore, we concur that a reasonably short iteration time would be beneficial to generate an initial solution. Then, the NLBA will be executed to find the first improvement, *firstImproveRHS,* based on the problem dimensions. If the considered problem is small, the improvement could occur by changing a small number of variables, this helps with exploitation inside small search subspaces. On the other hand, if the problem is too large, the first improvement may occur only after a change in a large number of variables. This varying of a large number of variables is what is called exploration within different search subspaces.

To find a reasonable, feasible initial solution, the black box solves the problem in the *iterationTime* without any interruption or additional constraints. Once the initial solution is found, that is considered as the *bestSoFarSolution* ($\bar{x}$) and the *bestSoFar* is updated by the initial objective value *iterationObj*. Afterward, the LBC is built based on the given initial solution *bestSoFarSolution* ($\bar{x}$). Importantly, the main while loop is iterated until one of the termination criteria is met. That is, until the elapsed processing time exceeds the *iterationTime* or the best lower bound becomes greater than or equal to the UB. During the main while loop, once the first improvement occurs in the objective value, compared to *bestSoFar*, the value of the right-hand side is saved as the *firstImproveRHS*. At the end of each iteration, the status of the given solution will be checked. Three different cases may arise for the iteration status, as follows:

(a) Optimal: since it is often a good point for exploiting the solution subspace, the time allotted to a certain iteration is set to a time limit *iterationTime*, then the algorithm checks the iteration objective *iterationObj*:

- If it improves the best so far, the value of *bestSoFar* is updated, the last LBC is deleted and rebuilt based on the reference solution *bestSoFarSolution* ($\bar{x}$), and the right-hand side value *currentRHS* is set equal to the *firstImproveRHS* to narrow the search subspace. Accordingly, at the next iteration, the algorithm is executed to exploit a small search subspace around a good point.
- If the iteration objective *iterationObj* cannot improve the *bestSoFar*, while the algorithm is still around a good point, the *currentRHS* is just increased by *increaseRateOfRHS*. We assume the previous search subspace is not large enough to contain better solutions. So, at the next iteration, the problem is solved again in a certain iteration time *iterationTime* but in a relatively larger search subspace. In other words, the algorithm explores a larger search subspace around a good point to find a better solution.

(b) Feasible: there are two reasons an iteration objective might be feasible, but not optimal:

(1) (1) Since the certain iteration time limit is not enough for searching the whole subspace, the algorithm cannot find the optimal solution, and/or

(2) (2) The search subspace is too small to contain an optimal solution.

In these cases, there are two alternatives:

- If the objective function of a given feasible solution *iterationObj* leads *bestSoFar* to improve, the *bestSoFar* is updated, the last LBC is deleted and

Fig. 2 Pseudocode for the adaptive parameter control structure of NLBA

Pseudocode for ANLBA

```
1   Initialize TL, iterationTime, initialRHS, increaseRateOfRHS,
    increaseRateOfTime;
2   Set iterationObjective = bestSoFar = +∞;
3   Create model
4   Solve the iteration zero;
5   Set bestSoFarSolution ($\bar{x}$)= zeroIterationSolution;
6   Update bestSoFar;
7   Set currentRHS = initialRHS;
8   Add the constraint $\Delta(x,\bar{x}) \ll currentRHS$to the model;
9   While (termination criteria not satisfied)
10    Solve the model (iterationTime, UB);
11    If (iterationStatus == "Optimal")
         a.  Set iterationTime by default;
12    Elseif (iterationStatus == "Feasible")
         a.  IterationTime * increaseRateOfTime;
13    Else
         a.  IterationTime *increaseRateOfTime ;
         b.  currentRHS * increaseRateOfRHS;
14    Endif;
15    If (iterationStatus == "Optimal" or iterationStatus == "Feasible")
16      If (iterationObj < bestSoFar)
17        If (numOfImproved == 1)
18        Set firstImproveRHS = currentRHS;
19        Endif
20      Update bestSoFar;
21      Update bestSoFarSolution ($\bar{x}$);
22      Set currentRHS = firstImproveRHS;
23      Delete the last constraint from the MIP model;
24      Add the constraint $\Delta(x,\bar{x}) \ll currentRHS$to the model;
25      Else
         a.  currentRHS * increaseRateOfRHS;
         b.  Update the last constraint with currentRHS;
26      Endif;
27    Endif;
28  Endwhile;
```

rebuilt based on the reference solution *bestSoFarSolution* ($\bar{x}$), and the certain iteration time limit *iterationTime* is just prolonged by *increaseRateOfTime*. So, the algorithm can exploit the current small search subspace in a relatively longer iteration time.

- If the objective function of a given feasible solution *iterationObj* fails to improve the *bestSoFar*, the certain iteration time limit *iterationTime*-, as well as the right-hand side value of the LBC currentRHS, are increased. In this case, the algorithm can explore a vast search subspace in a longer iteration time.

(c) Infeasible or Unknown: in this case, both the certain iteration time limit *iterationTime* and the right-hand side value of the LBC *currentRHS* are increased by *increaseRateOfTime* and *increaseRateOfRHS*, respectively, until the first feasible or optimal solution is found.

This adaptive structure for setting parameters interrupts the algorithm at each iteration to check the convergence speed, the status of the current solution, and the objective value, then cuts the search space according to this feedback. In ANLBA, the right-hand side value of the LBC will change based on some deterministic rules which help the algorithm adapt to the problem dimension. Additionally, this structure helps keep the algorithm from becoming trapped at a local optimum in a small search subspace, and helps it find better solutions with specific iteration time.

As mentioned previously, using exploitation and exploration is highly dependent on the characteristics of different problems, but the proposed adaptive parameter control structure can start searching within a small search subspace and try to find a better solution in a reasonable time. There are some key points in terms of the main contribution of this study that should be mentioned as a guideline about the proposed structure:

- This structure is not very sensitive to set the iteration time *iterationTime* since this iteration time is just used to abort the algorithm, check the status of the given solution, and decide for the next iteration. On the other hand, if the algorithm can find the optimal solution within a search subspace, the black box will be terminated. As a result, we recommend considering a short iteration time so that the algorithm will not be trapped in a local optimum.
- As we mentioned, considering first improvement *firstImproveRHS* after executing the NLBA could help us to adjust the adaptive structure to fit the problem. This parameter helps us to get information about problem dimensions, and when the algorithm finds a better solution, the adaptive structure sets the right-hand side value of the LBC *currentRHS* equal to *firstImproveRHS* to explore a small search subspace around a better given feasible solution.
- The rate of increase of the iteration time *increaseRateOfTime* and right-hand side value of the LBC *increaseRateOfRHS* are tools that can give the algorithm a chance of finding a better solution. These parameters should have smaller rather than larger values since the adaptive structure will be aborted and checked regularly to make decisions for the next iterations. More succinctly, the role of these parameters is jumping out from local optimums within small search subspaces.

In the following section, multi-commodity network design problems will be described in brief, and 26 problems are tested to compare the performance of the ANLBA with that of the original NLBA.

4 Formulation of the multi-commodity network design problems

Generally, there are several types of network design problems. In this section, the general model of the network design problem is presented [38].

Consider a network *G* = *(N, A)*, in which *N* is the set of nodes and *A* is the set of directed arcs. In this type of network problem, commodities can be distinct physical goods or the same goods with a different origin or destination. So *K* is the set of commodities, and d_k is the amount of goods which must flow from the origin *O(k)* to the desired destination *D(k)*. Each flow has fixed and variable cost components, so c_{ij}^k and F_{ij} are the per-unit arc routing cost of community *k* on the arc *(i, j)*, and the fixed arc design cost *(i, j)*, respectively. The general model can be written as follows [39]:

$$\min\left(\sum_{k\in K}\sum_{(i,j)\in A} c_{ij}^k f_{ij}^k + \sum_{(i,j)\in A} F_{ij}y_{ij}\right) \tag{7}$$

$$\sum_{j\in N} f_{ij}^k - \sum_{l\in N} f_{lj}^k = \begin{cases} d_k & ifi = O(k) \\ -d_k & ifi = D(k) \quad \forall K \in k \\ 0 & otherwise \end{cases} \tag{8}$$

$$\sum_{k\in K} f_{ij}^k \le K_{ij}y_{ij}\forall(i,j) \in A \tag{9}$$

$$(f, y) \in S \tag{10}$$

$$f_{ij}^k \ge 0, y_{ij} = 0 \; or \; 1\forall(i,j) \in A, \mathrm{k} \in \mathrm{K} \tag{11}$$

where y_{ij} and f_{ij}^k are the decision variables, y_{ij} is 0 if the arc (i,j) is closed and 1 if it is open, and f_{ij}^k is the amount of the commodity k which flows on the arc(i,j). The objective function of the model in Eq. (7) is to minimize the variable cost of commodities flowing on the arcs, as well as the fixed cost of the open arcs. Equation (8) is the balancing equation of the network flow problem. Equation (9) shows that the sum of the flows on each arc (i,j) must not exceed the maximum capacity of the arc.

5 Computational results

In this section, a total of 26 different multi-commodity network design problems are solved using 4 different methods for comparisons: the proposed ANLBA; the original NLBA; NLBA using Self-Adaptive Harmony Search (SAHS), which is described below; and CPLEX 12.6.2 – 64 Bit. Specifically, the program was written in the Java programming language and executed on a personal computer equipped with 8 GB of RAM and an Intel(R) Core(TM) i7-6500U CPU at 2.50 GHz, operating under Microsoft Windows 10. Furthermore, the termination criteria for all comparisons are a processing time of an hour and a half (5600 s) and/or five successive insignificant improvements in the objective function. Finally, the results are described in three subsections below, where the relative gaps of the objective function and processing time are presented in Tables 3, 4 and 5.

In Tables 3, 4 and 5, the PROB column indicates the characteristics of the problem, such as the number of nodes, arcs, and demands. The ANLBA columns refer to the values of the objective function of each problem as calculated by NLBA with the proposed adaptive parameter control structure along with the respective processing time and final status. Besides the stated termination criteria, the parameters of ANLBA were initialized as follows: The limited *iterationTime* set to 200 s., and the initial right-hand side value *initialRHS*, the increasing rate of right-hand side value *increaseRateOfRHS*, and the increasing rate of the iteration time *increaseRateOfTime* were all initialized at 2.

5.1 The NLBA in comparison to The ANLBA

Considering the stated termination criteria, the parameters of the NLBA are initialized as follows:, a certain zero iteration time *zeroIterationTime* was set to 200 s., the *initialRHS*, *increaseRateOfRHSForImprove*, and *increaseRateOfRHSForNotImprove* were initialized by the default values in the literature, 1, 1.1 and 2, respectively.

In Table 3, the NLBA columns refer to the objective value, the execution time, and the status of the final iteration given by NLBA for each problem, respectively. Finally, the GAP columns present the results of comparing the proposed ANLBA to the original NLBA. Overall relative gaps are reported along the last row. As Table 3 shows, the performance of the proposed ANLBA, as compared to the NLBA, improved by – 14.0% and – 663.53% with regards to the objective functions and processing times, respectively.

Table 3 Computational results of NLBA and ANLBA

Row	PROB	NLBA			ANLBA			Relative Gap %	
		Obj. Value	CPU time (s)	Final status	Obj. Value	CPU time (s)	Final status	Obj. value	CPU time (s)
1	20,230,200,V,L	94,213.00	5600.00	Time Reached	94,213.00	4225.00	Not improving	0.00	– 24.55
2	20,230,200,F,L	137,712.89	5600.00	Time reached	137,642.33	2674.00	Not improving	– 0.05	– 52.25
3	20,230,200,V,T	97,914.00	5600.00	Time reached	97,914.00	1895.00	Not improving	0.00	– 66.16
4	20,230,200,F,T	136,661.44	5600.00	Time reached	136,231.00	5600.00	Time reached	– 0.31	0.00
5	20,300,200,V,L	75,155.00	5600.00	Time reached	74,811.00	4151.00	Not improving	– 0.46	– 25.88
6	20,300,200,F,L	116,022.33	5600.00	Time reached	116,330.00	3827.00	Not improving	0.27	– 31.66
7	20,300,200,V,T	74,991.00	3456.00	Not improving	74,991.00	2292.00	Not improving	0.00	– 33.68
8	20,300,200,F,T	107,803.10	5600.00	Time reached	107,169.00	3987.00	Not improving	– 0.59	– 28.80
9	30,520,100,V,L	53,958.00	3544.00	Not improving	53,958.00	410.00	Not improving	0.00	– 88.43
10	30,520,100,F,L	94,809.56	5600.00	Time reached	94,043.00	5600.00	Time reached	– 0.81	0.00
11	30,520,100,V,T	52,046.00	5600.00	Time reached	52,046.00	3115.00	Not improving	0.00	– 44.38
12	30,520,100,F,T	99,394.56	5601.00	Time reached	98,076.00	5411.00	Time reached	– 1.33	– 3.39
13	30,520,400,V,L	112,909.64	5600.00	Time reached	112,961.44	5430.00	Time reached	0.05	– 3.04
14	30,520,400,F,L	152,756.78	5601.00	Time reached	150,254.86	5600.00	Time reached	– 1.64	– 0.02
15	30,520,400,V,T	114,672.69	5600.00	Time reached	114,640.46	5600.00	Time reached	– 0.03	0.00
16	30,520,400,F,T	157,682.14	5600.00	Time reached	153,453.22	5600.00	Time reached	– 2.68	0.00
17	30,700,100,F,L	60,121.67	5600.00	Time reached	60,143.00	2093.00	Not improving	0.04	– 62.63
18	30,700,100,V,T	45,871.50	5602.00	Time reached	45,891.00	1061.00	Not improving	0.04	– 81.06
19	30,700,100,F,T	54,975.50	5600.00	Time reached	55,014.00	3305.00	Not improving	0.07	– 40.98
20	30,700,400,V,L	98,561.11	5600.00	Time reached	98,670.33	5665.00	Time reached	0.11	1.16
21	30,700,400,F,L	142,128.93	5600.00	Time reached	136,954.36	5600.00	Time reached	– 3.64	0.00
22	30,700,400,V,T	95,673.56	5601.00	Time reached	95,656.00	5600.00	Time reached	– 0.02	– 0.02
23	30,700,400,F,T	132,666.22	5600.00	Time reached	130,974.88	5600.00	Time reached	– 1.27	0.00
24	100,400,10,F,T	64,113.00	5601.00	Time reached	63,764.00	5600.00	Time reached	– 0.54	– 0.02
25	100,400,30,F,L	49,018.00	5600.00	Time reached	49,115.00	3130.00	Not improving	0.20	– 44.11
26	100,400,30,F,T	139,478.33	5600.00	Time reached	137,514.00	3717.00	Not improving	– 1.41	– 33.63

Table 4 Computational results of NLBA coupled by SAHS and ANLBA

Row	PROB	NLBA coupled by SAHS			ANLBA			Relative Gap %	
		Obj. Value	CPU time (s)	Final status	Obj. Value	CPU time (s)	Final status	Obj. Value	CPU time (s)
1	20,230,200,V,L	94,213.00	5602.00	Time reached	94,213.00	4225.00	Not improving	0.00	− 24.58
2	20,230,200,F,L	138,533.67	3501.00	Not improving	137,642.33	2674.00	Not improving	− 0.64	− 23.62
3	20,230,200,V,T	97,914.00	3852.00	Not improving	97,914.00	1895.00	Not improving	0.00	− 50.80
4	20,230,200,F,T	136,252.00	5251.00	Time reached	136,231.00	5600.00	Time reached	− 0.02	6.65
5	20,300,200,V,L	75,219.00	5601.00	Time reached	74,811.00	4151.00	Not improving	− 0.54	− 25.89
6	20,300,200,F,L	116,291.00	4551.00	Not improving	116,330.00	3827.00	Not improving	0.03	− 15.91
7	20,300,200,V,T	74,991.00	2450.00	Not improving	74,991.00	2292.00	Not improving	0.00	− 6.45
8	20,300,200,F,T	107,465.80	3150.00	Not improving	107,169.00	3987.00	Not improving	− 0.28	26.57
9	30,520,100,V,L	53,958.00	2450.00	Not improving	53,958.00	410.00	Not improving	0.00	− 83.27
10	30,520,100,F,L	94,834.00	4551.00	Not improving	94,043.00	5600.00	Time reached	− 0.83	23.05
11	30,520,100,V,T	52,048.00	3151.00	Not improving	52,046.00	3115.00	Not improving	0.00	− 1.14
12	30,520,100,F,T	98,048.00	5601.00	Time reached	98,076.00	5411.00	Time reached	0.03	− 3.39
13	30,520,400,V,L	112,892.82	4901.00	Not improving	112,961.44	5430.00	Time reached	0.06	10.79
14	30,520,400,F,L	151,508.23	5604.00	Time reached	150,254.86	5600.00	Time reached	− 0.83	− 0.07
15	30,520,400,V,T	114,672.69	4550.00	Not improving	114,640.46	5600.00	Time reached	− 0.03	23.08
16	30,520,400,F,T	156,031.13	5603.00	Time reached	153,453.22	5600.00	Time reached	− 1.65	− 0.05
17	30,700,100,F,L	60,049.00	5600.00	Time reached	60,143.00	2093.00	Not improving	0.16	− 62.63
18	30,700,100,V,T	45,871.50	2801.00	Not improving	45,891.00	1061.00	Not improving	0.04	− 62.12
19	30,700,100,F,T	55,014.00	2451.00	Not improving	55,014.00	3305.00	Not improving	0.00	34.84
20	30,700,400,V,L	98,410.00	5601.00	Time reached	98,670.33	5665.00	Time reached	0.26	1.14
21	30,700,400,F,L	139,184.00	5605.00	Time reached	136,954.36	5600.00	Time reached	− 1.60	− 0.09
22	30,700,400,V,T	95,894.49	5604.00	Time reached	95,656.00	5600.00	Time reached	− 0.25	− 0.07
23	30,700,400,F,T	132,216.83	5603.00	Time reached	130,974.88	5600.00	Time reached	− 0.94	− 0.05
24	100,400,10,F,T	64,207.00	3851.00	Not improving	63,764.00	5600.00	Time reached	− 0.69	45.42
25	100,400,30,F,L	49,115.00	4900.00	Not improving	49,115.00	3130.00	Not improving	0.00	− 36.12
26	100,400,30,F,T	139,243.00	3850.00	Not improving	137,514.00	3717.00	Not improving	− 1.24	− 3.45

5.2 The hybrid structure of NLBA and SAHS in comparison to the ANLBA

In order to evaluate the performance of the proposed ANLBA, we decided to use a heuristic optimization algorithm named Self-Adaptive Harmony Search (SAHS), which includes an adaptive structure to control the parameters of NLBA. More generally, Harmony Search (HS) is a heuristic algorithm which consists of a set of solution vectors governed by three rules including random selection, memory consideration and pitch adjustment which make for a balance between exploration and exploitation by selecting a value from the possible range of variables [40]. Recently, the HS algorithm has captured much attention, and various extensions of it have been proposed and applied to solve a wide range of optimization problems [41–47]. Furthermore, different variants of the HS algorithm employ novel methods for generating new solution vectors that enhance accuracy and convergence rate. As a result of good performance and convergence in parameter tuning, the HS algorithm and its extensions are widely used as parameter tuning approaches for improving structural optimization and dynamic parameter adaptation [48–53].

Here, we used the SAHS algorithm as a parameter tuning method and applied this adaptive structure for setting parameter values of the NLBA. In the following subsection, the HS and SAHS algorithms are briefly described. Then, the framework of the hybrid structure consisting of the SAHS algorithm and the NLBA, as well as the results of this hybrid structure as compared to the proposed ANLBA are presented.

5.2.1 The general procedure of HS algorithm

Step 1: An initial population of n-dimension harmony vectors are randomly generated and stored in a Harmony Memory (HM).
Step 2: A new harmony is generated from all of the solutions in the HM based on a memory consideration rule, a pitch adjustment rule, and a random re-initialization.
Step 3: The new candidate vector will be compared to the worst harmony vector in the HM. If the new candi-

Table 5 Computational results of CPLEX and ANLBA

Row	PROB	CPLEX			ANLBA			Relative Gap %	
		Obj. Value	CPU time (s)	Final status	Obj. Value	CPU time (s)	Final status	Obj. value	CPU time (s)
1	20,230,200,V,L	94,254.00	5600.12	Time reached	94,213.00	4225.00	Not improving	− 0.04	− 24.56
2	20,230,200,F,L	137,854.00	5600.15	Time reached	137,642.33	2674.00	Not improving	− 0.15	− 52.25
3	20,230,200,V,T	97,914.00	2529.87	Optimal found	97,914.00	1895.00	Not improving	0.00	− 25.09
4	20,230,200,F,T	136,225.00	5600.15	Time reached	136,231.00	5600.00	Time reached	0.00	0.00
5	20,300,200,V,L	74,811.00	5600.36	Time reached	74,811.00	4151.00	Not improving	0.00	− 25.88
6	20,300,200,F,L	116,186.00	5598.73	Time reached	116,330.00	3827.00	Not improving	0.12	− 31.65
7	20,300,200,V,T	74,991.00	1583.73	Optimal found	74,991.00	2292.00	Not improving	0.00	44.72
8	20,300,200,F,T	107,314.00	5600.39	Time reached	107,169.00	3987.00	Not improving	− 0.14	− 28.81
9	30,520,100,V,L	53,958.00	479.09	Optimal found	53,958.00	410.00	Not improving	0.00	− 14.42
10	30,520,100,F,L	93,967.00	5600.17	Time reached	94,043.00	5600.00	Time reached	0.08	0.00
11	30,520,100,V,T	52,046.00	3672.24	Optimal found	52,046.00	3115.00	Not improving	0.00	− 15.17
12	30,520,100,F,T	97,259.00	5600.11	Time reached	98,076.00	5411.00	Time reached	0.84	− 3.38
13	30,520,400,V,L	112,923.21	5600.24	Time reached	112,961.44	5430.00	Time reached	0.03	− 3.04
14	30,520,400,F,L	149,835.25	5600.32	Time reached	150,254.86	5600.00	Time reached	0.28	− 0.01
15	30,520,400,V,T	114,676.64	5600.35	Time reached	114,640.46	5600.00	Time reached	− 0.03	− 0.01
16	30,520,400,F,T	153,884.11	5600.51	Time reached	153,453.22	5600.00	Time reached	− 0.28	− 0.01
17	30,700,100,F,L	59,958.00	5600.12	Time reached	60,143.00	2093.00	Not improving	0.31	− 62.63
18	30,700,100,V,T	45,871.50	5601.67	Time reached	45,891.00	1061.00	Not improving	0.04	− 81.06
19	30,700,100,F,T	54,904.00	5600.31	Time reached	55,014.00	3305.00	Not improving	0.20	− -40.99
20	30,700,400,V,L	97,984.00	5600.26	Time reached	98,670.33	5665.00	Time reached	0.70	1.16
21	30,700,400,F,L	146,667.04	5600.34	Time reached	136,954.36	5600.00	Time reached	− 6.62	− 0.01
22	30,700,400,V,T	95,315.60	5600.35	Time reached	95,656.00	5600.00	Time reached	0.36	− 0.01
23	30,700,400,F,T	131,217.04	5600.35	Time reached	130,974.88	5600.00	Time reached	− 0.18	− 0.01
24	100,400,10,F,T	63,753.00	5600.09	Time reached	63,764.00	5600.00	Time reached	0.02	0.00
25	100,400,30,F,L	49,115.00	5600.03	Time reached	49,115.00	3130.00	Not improving	0.00	− 44.11
26	100,400,30,F,T	136,435.00	5600.05	Time reached	137,514.00	3717.00	Not improving	0.79	− 33.63

date vector is better than the worst vector in the HM, the worst one is replaced by the new candidate vector and HM is updated.
Step 4: This process is repeated until a termination criterion is met.

Assume LB and UB are the lower bound and upper bound of the possible range of variables, HMS is the memory size, HMCR is the rate of choosing from the memory, PAR is the pitch adjustment rate, and BW is the distance bandwidth. The process of improvising a new harmony vector is shown in Fig. 3.

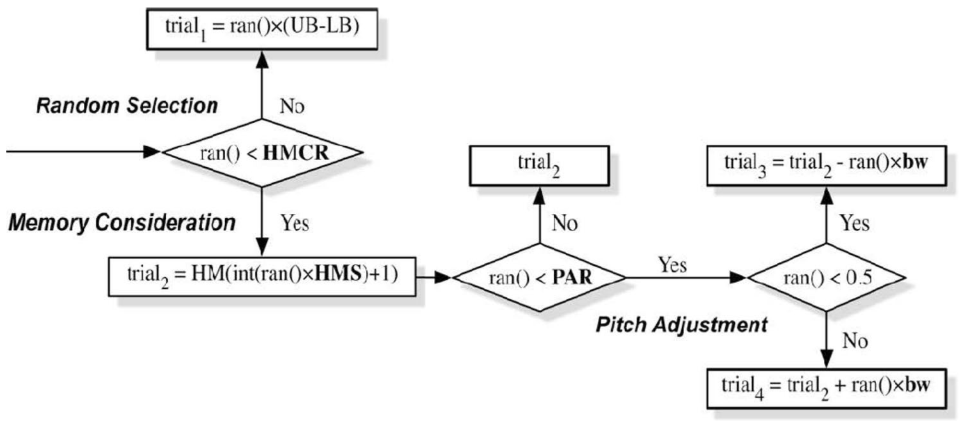
trial$_1$ = ran()×(UB-LB)
Random Selection
No
ran() < HMCR
Memory Consideration
Yes
trial$_2$ = HM(int(ran()×HMS)+1)
trial$_2$
No
ran() < PAR
Yes
Pitch Adjustment
trial$_3$ = trial$_2$ - ran()×bw
Yes
ran() < 0.5
No
trial$_4$ = trial$_2$ + ran()×bw

Fig. 3 Improvise a new harmony vector at each iteration in HS algorithm

The parameter HMCR, which varies between 0 and 1, controls the balance between exploration and exploitation. The HMCR value determines the random selection from the possible range of variables, or the memory consideration to pick one pitch from HM. Once one pitch has been picked from the HM, the PAR determines whether or not further adjustment is required according to a variable-distance bandwidth BW. This pitch adjustment step is similar to a local search mechanism that finds a new harmony from outside of the HM [40].

5.2.2 The general framework of the SAHS algorithm

Because PAR and BW in the HS algorithm have a great influence on the quality of final solutions and control the convergence rate and ability for fine-tuning, [42] proposed a modified variant of HS, called SAHS, which includes a self-adaptive mechanism to improve the pitch adjustment step of HS using Eq. (12) and Eq. (13).

Let min (HM_i) and max (HM_i) denote the lowest and the highest values of the ith variable in the HM, then the selected pitch from the HM called a trial is adjusted as follows:

$$trial_i + [\max(HM_i) - trial_i] \times \text{ran}[0,1] \tag{12}$$

$$trial_i - [trial_i - \min(HM_i)] \times \text{ran}[0,1] \tag{13}$$

The pitch adjustment through this mechanism is based on the LB and UB of the variables and it would not violate the boundary constraint of variables.

5.2.3 The hybrid structure consists of the NLBA coupled with the SAHS algorithm

In this structure, we consider *increaseRateOfRHSForImprove,* and *increaseRateOfRHSForNotImprove* as parameters for adjusting by SAHS algorithm. So, the HM consists of solution vectors of two dimensions, including *increaseRateOfRHSForImprove*, and *increaseRateOfRHSForNotImprove* which were randomly generated in the range [1, 4]. According to the literature, the values of HMS and HMCR were set to 50 and 0.99, respectively. Also, a certain zero iteration time, *zeroIterationTime,* and *initialRHS* were set to 200 s. and 1, as stated in Sect. 5.1, respectively.

As mentioned, *TL*, *zeroIterationTime*, *HMS, HMCR* were initialized, so the NLBA is executed to solve the zero iteration and update the *bestSoFarSolution* (x). Then, the main while loop of the NLBA (presented in Fig. 1) is iterated until one of the termination criteria is met. At each iteration, before calling the black box, a new harmony vector is improvised according to the general framework of the HS algorithm for random selection, memory consideration, and the pitch adjustment mechanism of the SAHS algorithm stated in Eq. (12) and Eq. (13). The results of the comparison between the proposed ANLBA and this hybrid structure is presented in Table 4. As the overall relative gaps show, the proposed ANLBA, when compared to the hybrid structure consisting of the NLBA and SAHS algorithm, improved by − 8.96% and − 228.17% in relative gaps of the objective functions and processing times, respectively.

5.3 The proposed ANLBA in comparison to CPLEX

In Table 5, the result of the problems solved by the proposed ANLBA and CPLEX are reported. The CPLEX columns show the objective function, the CPU time, and the status of the final iteration. The results confirm that the proposed ANLBA outperformed CPLEX in terms of objective function values by − 3.67%, and in terms of processing times, by − 440.85%.

In sum, the algorithm using the proposed adaptive parameter control structure was found to have better solution accuracy and faster processing times than the original NLBA, the NLBA coupled with SAHS, and even CPLEX.

6 Conclusion and discussion

Given the paramount importance of parameter setting and parameter control in optimization algorithms, an adaptive parameter control structure was proposed to enhance the performance of the NLBA using CPLEX as a black box, to solve MIP problems. The proposed adaptive parameter control structure regularly aborts the algorithm checking convergence speed and instantaneous dimensions to determine the search direction at the next iteration. In other words, the proposed adaptive parameter control structure proposes some deterministic rules in a framework to enhance exploitation and exploration. Additionally, this structure will check the status and the value of a given feasible solution whenever termination criteria are met, and the right-hand side value of the local constraint and the certain iteration time are changed according to the deterministic rules.

Once the parameter control structure had been described, its excellent performance was demonstrated on a series of computational experiments regarding 26 multi-commodity network design problems. Indeed, the proposed structure improved the quality of the objective function value in a shorter processing time, compared to the original NLBA, the NLBA coupled with SAHS, and even CPLEX.

More precisely, improvements the proposed ANLBA made in objective function values and compared to the other algorithms were as follows: – 14.01% for the original NLBA, – 8.96% for the NLBA coupled with SAHS, and – 3.67% for CPLEX. Similarly improvements in processing times were shown to be – 663.53% compared to the original NLBA, – 228.17% for the NLBA-SAHS hybrid structure, and – 440.85% compared to CPLEX. The results confirmed that significant time and money savings can be obtained by solving MIP problems with the proposed adaptive parameter control structure to develop a network design plan.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Karafotias G, Hoogendoorn M, Eiben ÁE (2015) Parameter control in evolutionary algorithms: trends and challenges. IEEE Trans Evol Comput 19:167–187
2. Dobslaw F (2010) Recent development in automatic parameter tuning for metaheuristics. In: Proceedings of the 19th annual conference of doctoral students-WDS 2010, 2010
3. Eiben AE, Hinterding R, Michalewicz Z (1999) Parameter control in evolutionary algorithms. Evol Comput IEEE Trans 3:124–141
4. Lin L, Gen M (2009) Auto-tuning strategy for evolutionary algorithms: balancing between exploration and exploitation. Soft Comput 13:157–168
5. Tvrdık J (2009) Self-adaptive variants of differential evolution with exponential crossover. Anal West Univ Timisoara Ser Math-Inform 47:151–168
6. Zamuda A, Brest J (2015) Self-adaptive control parameters' randomization frequency and propagations in differential evolution. Swarm Evol Comput 25:72–99
7. Fan Q, Yan X (2015) Self-adaptive differential evolution algorithm with discrete mutation control parameters. Expert Syst Appl 42:1551–1572
8. Guo Z, Liu G, Li D, Wang S (2017) Self-adaptive differential evolution with global neighborhood search. Soft Comput 21:3759–3768
9. Fan Q, Yan X (2016) Self-adaptive differential evolution algorithm with zoning evolution of control parameters and adaptive mutation strategies. IEEE Trans Cybern 46:219–232
10. Wang Y, Li B, Weise T, Wang J, Yuan B, Tian Q (2011) Self-adaptive learning based particle swarm optimization. Inf Sci 181:4515–4538
11. Xue Y, Jiang J, Zhao B, Ma T (2017) A self-adaptive artificial bee colony algorithm based on global best for global optimization. Soft Comput 22:1–18
12. Hutter F, Stuetzle T, Leyton-Brown K, Hoos HH (2014) ParamILS: an automatic algorithm configuration framework. arXiv preprint arXiv:1401.3492
13. Precup R-E, David R-C, Petriu EM, Preitl S, Rădac M-B (2014) Novel adaptive charged system search algorithm for optimal tuning of fuzzy controllers. Expert Syst Appl 41:1168–1175
14. Jansen T, De Jong KA, Wegener I (2005) On the choice of the offspring population size in evolutionary algorithms. Evol Comput 13:413–440
15. Lagos C, Crawford B, Soto R, Cabrera E, Vega J, Johnson F et al (2016) Improving tabu search performance by means of automatic parameter tuning. Can J Electr Comput Eng 39:51–58
16. Wang H, Zhou X, Sun H, Yu X, Zhao J, Zhang H et al (2017) Firefly algorithm with adaptive control parameters. Soft Comput 21:5091–5102
17. Castillo O, Soto C, Valdez F (2018) A review of fuzzy and mathematic methods for dynamic parameter adaptation in the firefly algorithm. In: advances in data analysis with computational intelligence methods, Springer, 2018, pp 311–321
18. Deng W, Xu J, Zhao H (2019) An improved ant colony optimization algorithm based on hybrid strategies for scheduling problem. IEEE Access 7:20281–20292
19. Yang X-S, Deb S, Hanne T, He X (2019) Attraction and diffusion in nature-inspired optimization algorithms. Neural Comput Appl 31:1987–1994
20. Nseef SK, Abdullah S, Turky A, Kendall G (2016) An adaptive multi-population artificial bee colony algorithm for dynamic optimisation problems. Knowl-Based Syst 104:14–23
21. Xia Y-M, Chen J-l, Meng X-W (2008) On the dynamic ant colony algorithm optimization based on multi-pheromones. In: Seventh IEEE/ACIS international conference on computer and information science, 2008. ICIS 08, pp. 630–635
22. Castillo O, Neyoy H, Soria J, Melin P, Valdez F (2015) A new approach for dynamic fuzzy logic parameter tuning in ant colony optimization and its application in fuzzy control of a mobile robot. Appl Soft Comput 28:150–159
23. Deng W, Zhao H, Zou L, Li G, Yang X, Wu D (2017) A novel collaborative optimization algorithm in solving complex optimization problems. Soft Comput 21:4387–4398
24. Liu Y, Lu H, Cheng S, Shi Y (2019) an adaptive online parameter control algorithm for particle swarm optimization based on reinforcement learning. IEEE Congress Evol Comput (CEC) 2019:815–822
25. Li D, Guo W, Wang L (2019) Niching particle swarm optimizer with entropy-based exploration strategy for global optimization. In: International conference on swarm intelligence, 2019, pp 118–127
26. Son B, Kim J-S, Kim J-W, Kim Y-J, Jung S-Y (2019) Adaptive particle swarm optimization based on kernel support vector machine for optimal design of synchronous reluctance motor. IEEE Trans Magn 55:1–5
27. Dong W, Zhou M (2017) A supervised learning and control method to improve particle swarm optimization algorithms. IEEE Trans Syst Man Cybern Syst 47:1135–1148
28. Song Z, Liu B, Cheng H (2019) Adaptive particle swarm optimization with population diversity control and its application in tandem blade optimization. Proc Inst Mech Eng Part C J Mech Eng Sci 233:1859–1875
29. Aoun O, Sarhani M, El Afia A (2018) Hidden markov model classifier for the adaptive particle swarm optimization. In: Recent developments in metaheuristics, Springer, 2018, pp 1–15
30. Xu Y, Chen H, Luo J, Zhang Q, Jiao S, Zhang X (2019) Enhanced Moth-flame optimizer with mutation strategy for global optimization. Inf Sci 492:181–203
31. Xu Y, Chen H, Heidari AA, Luo J, Zhang Q, Zhao X et al (2019) An efficient chaotic mutative moth-flame-inspired optimizer for global optimization tasks. Expert Syst Appl 129:135–155
32. Nasser AB, Zamli KZ (2018) Parameter free flower algorithm based strategy for pairwise testing. In: Proceedings of the 2018 7th international conference on software and computer applications, 2018, pp 46–50
33. Nalepa J, Blocho M (2016) Adaptive memetic algorithm for minimizing distance in the vehicle routing problem with time windows. Soft Comput 20:2309–2327

34. H. A. H. A. f. S. M. I. P. Pooladi, Solving a Railway Model as a Case study. MSc Thesis, Iran University of Science and Technology
35. Fischetti M, Lodi A (2003) Local branching. Math Program 98:23–47
36. Fallahia M, Amiri S, Yaghinic M (2014) A parameter tuning methodology for metaheuristics based on design of experiments. Int J Eng Technol 2:497–521
37. Yaghini M, Momeni M, Sarmadi M (2013) An improved local branching approach for train formation planning. Appl Math Model 37:2300–2307
38. Magnanti TL, Wong RT (1984) Network design and transportation planning: models and algorithms. Transp Sci 18:1–55
39. Yaghini M, Akhavan R (2012) Multicommodity network design problem in rail freight transportation planning. Proc-Soc Behav Sci 43:728–739
40. Geem ZW, Kim JH, Loganathan GV (2001) A new heuristic optimization algorithm: harmony search. Simulation 76:60–68
41. Pan Q-K, Suganthan PN, Tasgetiren MF, Liang JJ (2010) A self-adaptive global best harmony search algorithm for continuous optimization problems. Appl Math Comput 216:830–848
42. Wang C-M, Huang Y-F (2010) Self-adaptive harmony search algorithm for optimization. Expert Syst Appl 37:2826–2837
43. Kattan A, Abdullah R (2013) A dynamic self-adaptive harmony search algorithm for continuous optimization problems. Appl Math Comput 219:8542–8567
44. Wang L, Hu H, Liu R, Zhou X (2019) An improved differential harmony search algorithm for function optimization problems. Soft Comput 23:4827–4852
45. Omran MG, Mahdavi M (2008) Global-best harmony search. Appl Math Comput 198:643–656
46. El-Abd M (2013) An improved global-best harmony search algorithm. Appl Math Comput 222:94–106
47. Mahdavi M, Fesanghary M, Damangir E (2007) An improved harmony search algorithm for solving optimization problems. Appl Math Comput 188:1567–1579
48. Peraza C, Valdez F, Castro JR, Castillo O (2018) Fuzzy dynamic parameter adaptation in the harmony search algorithm for the optimization of the ball and beam controller. Adv Oper Res 2018:1–16
49. Dash R (2018) An adaptive harmony search approach for gene selection and classification of high dimensional medical data. J King Saud University-Comput Inf Sci (in press)
50. Choi YH, Lee HM, Yoo DG, Kim JH (2018) Improvement of search efficiency in optimization algorithm using self-adaptive harmony search algorithms. J Korea Academia-Ind Cooper Soc 19:1–11
51. Castillo O, Melin P, Valdez F, Soria J, Ontiveros-Robles E, Peraza C et al (2019) Shadowed type-2 fuzzy systems for dynamic parameter adaptation in harmony search and differential evolution algorithms. Algorithms 12:17
52. Zhao F, Liu Y, Zhang C, Wang J (2015) A self-adaptive harmony PSO search algorithm and its performance analysis. Expert Syst Appl 42:7436–7455
53. Hasançebi O, Erdal F, Saka MP (2009) Adaptive harmony search method for structural optimization. J Struct Eng 136:419–431