*Journal Computer Science*

Original Research Paper

# Adaptive Acceptance Criterion (AAC) Algorithm for Optimization Problems

**Anmar Abuhamdah**

*Department of Information Systems,*
*College of Computer Science and Engineering, Taibah University, Madinah, KSA*

*Article history*
Received: 19-12-2014
Revised: 4-7-2015
Accepted: 7-7-2015

Email: aabuhamdah@taibahu.edu.sa

**Abstract:** Optimization methods commonly are designed for solving the optimization problems. Local search algorithms are optimization method, which are good candidate in exploiting the search space. However, most of them need parameter tuning and incapable of escaping from local optima. This work proposes non-parametric Acceptance Criterion (AC) that not relies on user-defined, which motivate to propose an Adaptive Acceptance Criterion (AAC). AC accepts a little worse solution based on comparing the candidate and best solutions found values to a stored value. The value is stored based on the lowest value of comparing the candidate and best solution found, when a new best solution found. AAC adaptively escape from local optima by employing a similar diversification idea of a previous proposed (ARDA) algorithm. In AAC, an estimated value added to the threshold (when the search is idle) to increase the search exploration. The estimated value is generated based on the frequency of the solutions quality differences, which are stored in an array. The progress of the search diversity is governed by the stored value. Six medical benchmark datasets for clustering problem (which are available in UCI Machine Learning Repository) and eleven benchmark datasets for university course timetabling problems (Socha benchmark datasets) are used as test domains. In order to evaluate the effectiveness of the propose AAC, comparison made between AC, AAC and other approaches drawn from the scientific literature. Results indicate that, AAC algorithm is able to produce good quality solutions which are comparable to other approaches in the literature.

**Keywords:** Local Search Algorithms, Adaptive Acceptance Criterion, Medical Clustering Problems, Multi K-Means, Course Timetabling Problem

## Introduction

In optimization problems, there are various approaches inspired from a number of scientific disciplines like artificial intelligence, computational intelligence and operations research (Tripathy, 1980) in solving different optimization problems (i.e., scheduling and clustering). These approaches are classified into several categorizes (i.e. sequential, cluster, generalized search (meta-heuristic) and constraint-based methods) (Carter and Laporte, 1998), which re-categorized in different categorizes (Lewis, 2008).

However, some of these algorithms are capable of producing good quality solutions and some perform poorly. Moreover, they need investigation to tune their parameters. Meanwhile, the successful methods usually maintain adaptive criterion, intelligent neigh borhoods selection and hybridization.

In recent years, the researchers investigate the approaches toward enhancing the performance of the previous methods in the literature by hybridizing different acceptance criteria, which produce a complex methods for solving the optimization problems and needs more parameter tuning. The complexity of the problems needs to finding simple methods which employs an acceptance criterion that not relies on user-defined (non-parameterized acceptance criterion), or find a mechanism that intensify and diversify the search to produce good quality solutions and may adaptively get out from local optima.

Local Search (LS) Method is a simple meta-heuristics approaches widely used for solving the optimization problems by searching from current solution to its neighbor solutions (Pirlot, 1996). LS use the single based for solving computationally hard optimization problems. LS can be used for maximizing or minimizing the

solution quality (Schaerf, 1999). LS consist of three important entities, the search space, neighborhood relation and the objective function (which evaluates the solution quality) (Schaerf, 1999).

The strength of the LS algorithm is in exploiting the search space (or the intensification process) (Ayvaz *et al*., 2012). However, the disadvantage of the LS approaches (Ayvaz *et al*., 2012) and the descent heuristic techniques (Schaerf, 1999) is that they are incapable of escaping local optima (minima), which is the strength of the population based approaches (Ayvaz *et al*., 2012). However, the disadvantage of the LS approaches is referring to their acceptance criterions, which not employ an adaptive mechanism to escape from local optima and almost of them needs to tune their parameters. Therefore, in this study, non-parametric Acceptance Criterion (AC) is proposed for optimization problems to overcome the disadvantage of the LS h approaches in tuning the parameters, which motivates to increase the diversification strategy (or exploring the search space) by proposing an Adaptive Acceptance Criterion (AAC) to overcome the disadvantage of the LS approaches in their incapability of escaping from the local optima. AC acceptance criterion relies on the comparison between the candidate and best solutions with stored value. The stored value generated based on an improvement on the best solution. AAC employ a similar idea of the diversification in a previous proposed algorithm, which add an estimated value to the threshold (when the search idle) to increase the diversification. The estimated value is generated based on the repetition of the solutions quality differences that are stored in an array.

The aim of this work is to propose an Adaptive Acceptance Criterion (AAC) for better diversification strategy for the optimization problems in order to produce a good solution quality.

Two difference test domains are used (i.e. clustering and timetabling problems). The first test domain is a medical clustering problem, which is partitioning a set of objects into a number of clusters of similar characteristic (Brucker, 1978). A cluster is a collection of similar objects and dissimilar to the other objects in other clusters (Brucker, 1978). The similarity of a cluster is classified based on certain object function (or distance function) (Saha *et al*., 2010). Partitioning a set of objects into two or more clusters is an NP-hard problem for it is difficulty in finding an optimal partition in reasonable time (Dasgupta and Freund, 2009). Finding a good clustering partitions or near to optimal are depends on the problem representation and the methodology to partitions the clusters during the search (Jain *et al*., 1999). The methodologies (or algorithms) are used to generate the initial clusters partitions. The initial cluster partitions quality (which termed as the minimal distance value) is calculated by using a distance function. In clustering problem, there are a numerous approaches used to generate the initial cluster partitions (or minimal distance value) (Holland, 1975) such as, *Multi K-Means* algorithm (Davidson and Satyanarayana, 2003), *Fuzzy C-Means* algorithm (Hong, 2006) and others (Berry and Linoff, 1997). Then, the minimal distance value iteratively improve by any algorithm (such as local search algorithms) to produce good quality clusters. The second test domain is the university course timetabling problems, which involves assigning a set of courses (events) and students to a fixed number of rooms and timeslots subject to a variety of constraints (Petrovic and Burke, 2004). Constraints in a timetabling problem can be classified as *hard* and *soft* constraints (Petrovic and Burke, 2004). The goal of solving timetabling problems is to satisfy all the hard constraints and attempt to accommodate the soft constraints as much as possible (in order to produce a good-quality timetable). All hard constraints must be satisfied in order to obtain a feasible timetable, whilst soft constraints can be accommodate and violated if necessary, where each violated constraint is penalized. The smaller value of penalizing overall penalty values is a better quality of the timetable. University course timetabling problems have been classified as an NP-hard problem; therefore it is difficult (in general) to find an optimal solution (for larger size instances) in a reasonable time (Schaerf, 1999). Finding good quality solutions to these problems be subject to the approach used and the problem representation employed during the search (Schaerf, 1999).

In recent years there are several approaches (or algorithms) used in both clustering and university timetabling problems to improve the solution quality. However, both Simulated Annealing (SA) (Kittaneh *et al*., 2012; Abuhamdah and Ayob, 2009) and Great Deluge algorithm (GD) (Abuhamdah and Ayob, 2009; Abuhamdah, 2012) are applied in the both test domain (i.e. medical clustering and university course timetabling problems), in addition that, they are LS methods widely used for their good performance. Therefore, in order to evaluate the performance of AAC algorithm, six benchmark datasets for medical clustering problems (which available in UCI Machine Learning Repository) and eleven benchmark datasets for university course timetabling (Socha benchmark datasets) are used, to compare the performance between AC, AAC, SA, GD and other approaches drawn from the scientific literature. Results demonstrate that AAC is able to produce statistically significantly higher quality solutions, outperforming many other LS approaches like SA and GD and obtain good quality solutions with other approaches on both domains (the medical clustering and course timetabling) datasets performances in line with other researchers.

This paper is structured as follows: the problem description is presented in section 2. Section 3 describes the methodology. AAC algorithm is presented in section

4. section 5 discusses the experimental results. Finally, section 6 presents the conclusion.

## Problem Description

In this study, two problem domains are used. The first problem domain is a medical clustering problem described in section 2.1, while the second problem domain is university course timetabling problem described in section 2.2.

### *Medical Clustering Problem*

In this problem domain, six benchmark datasets are used tackle the medical clustering problems that denoted for research purpose, which available in UCI machine learning repository (http://archive.ics.uci.edu/ml/index.html). These datasets are information about the diseases and were collected from real infected patients. These datasets are chosen with difference number of patterns and different complexity as summarized in Table 1. Each dataset is available with a fixed number of clusters for research purpose. Note that, Dataset 1 is Haberman's Survival Database (*H.S*), Dataset 2 is BUPA Liver Disorders Database (*B.L.D*), Dataset 3 is Pima Indian Diabetes Database (*P.I.D*), Dataset 4 is Wisconsin Breast Cancer Database (*B.C*), Dataset 5 is Thyroid gland data Disease Database (*T.D*) and Dataset 6 is Lung Cancer Database (*L.C)*. For example in Table 1, the dataset number 6 is Lung Cancer Database (L.C) have 56 attributes with 32 integer instances and categorized as three clusters in the initial clusters partition, the first cluster takes 9 instances, the second cluster takes 13 instances and the third cluster takes 10 instances.

The initial cluster quality for each dataset can be evaluated by using a distance function to calculate the minimal distance value. The distance function value can evaluate the algorithm performance (Maulik and Bandyopadhyay, 2000). Where, the smallest value indicates better clusters quality (or minimal distance value).

Euclidean Distance as illustrated in Equation 1 (Wang, 2007) is a distance function widely used for calculating the minimal distance value, where it performs well when the clusters are isolated and compact (Zhang, 2001). For example, assume there is a dataset $X = \{x1, x2, \ldots xn\}$ with $n$ objects and we need to cluster it into $K$ number of clusters, where $i$ and $j$ are two of n-dimensional data objects:

$$d(i,j) = \sqrt{\sum_{i=1}^{n}\left(x_i - x_j\right)^2} \quad (1)$$

In this study, there are two different ways for calculating the minimal distance values are used, as follows:

### *Between Objects Distance Function Value*

In this calculation, the minimal distance value is calculated based on distance between each data pattern and the pattern next to it. The idea of this calculation is to minimize the distance between the data patterns themselves in the same cluster (Wang, 2007).

### *Between Centers Distance Function Value*

In this calculation, the minimal distance value is calculated based on the distance between each data pattern and their cluster center that it belongs to it (Wang, 2007), where a new center is calculated. The idea of this calculation is minimize the distance between the data patterns and their centers.

### *University Course Timetabling Problem*

In this problem domain, eleven standard benchmark datasets were introduced by Socha *et al*. (2002) are used, which seek to optimize the students' satisfaction for the university course timetabling problem. The problem consists of:

- A set of Rooms *R* in which events can take place
- A set of Events (courses) *E* to be scheduled in 45 timeslots (5 days of 9 hours each and one hour for each timeslot)
- A set of features *F* characterize the rooms
- A set of Students *S* who attend the events

These datasets are categorized into three groups: small (i.e. *small 1, small 2, small 3, small 4* and *small 5*), medium (i.e. *medium 1, medium 2, medium 3, medium 4* and *medium 5*) and large (*large*) datasets (see Table 2 for more detailed description).

Table 2 also shows the number of students, events, rooms and features as well as the conflict density (*CD*) for each dataset (representing the complexity), an approximation of the number of students enrolled in each event (Students/ Events) and an approximation of the number of available rooms for each event (Rooms/Events) which are calculated as in (Chiarandini *et al*., 2006).

These datasets have three hard constraints (H1, H2 and H3) and three soft constraints (S1, S2 and S3), as follows:

### *Hard Constraints*

H1: No student attends more than one event at the same time
H2: The room has to be large enough for all the attending students and has all the features required by the event
H3: Only one event takes place in each room in any timeslot

### *Soft Constraints*

S1: A student should not have a class in the last timeslot of the day.
S2: A student should not have more than two classes consecutively.
S3: A student should not have a single class on a day.

Table 1. Six Benchmark datasets for medical clustering problems

| Dataset number | Dataset name | No. of attributes | No. of instances | Attributes type | No. of clusters | Clusters distribution |
|---|---|---|---|---|---|---|
| | *H.S* | 4 | 306 | Integer | 2 | 225 and 81 |
| | *B.L.D* | 7 | 345 | integer, categorical and real | 2 | 145 and 200 |
| | *P.I.D* | 8 | 768 | integer and real | 2 | 500 and 268 |
| | *B.C* | 10 | 699 | Integers | 3 | 123, 240 and 363 |
| | *T.D* | 21 | 215 | categorical and real | 3 | 150, 30 and 35 |
| | *L.C* | 56 | 32 | Integers | 3 | 9, 13 and 10 |

Table 2. The course timetabling datasets

| Dataset | # Students | # Events | # Rooms | # Features | CD | Students/Events | Rooms/Events |
|---|---|---|---|---|---|---|---|
| Small 1 | 80 | 100 | 5 | 5 | 10.96 | 4.98 | 0.82 |
| Small 2 | 80 | 100 | 5 | 5 | 13.92 | 5.36 | 0.79 |
| Small 3 | 80 | 100 | 5 | 5 | 9.71 | 4.65 | 1.00 |
| Small 4 | 80 | 100 | 5 | 5 | 7.16 | 3.45 | 1.39 |
| Small 5 | 80 | 100 | 5 | 5 | 15.10 | 5.99 | 1.17 |
| Medium 1 | 200 | 400 | 10 | 5 | 37.38 | 8.85 | 2.23 |
| Medium 2 | 200 | 400 | 10 | 5 | 37.66 | 8.84 | 1.91 |
| Medium 3 | 200 | 400 | 10 | 5 | 40.44 | 8.85 | 1.91 |
| Medium 4 | 200 | 400 | 10 | 5 | 37.50 | 8.81 | 1.88 |
| Medium 5 | 200 | 400 | 10 | 5 | 28.27 | 8.66 | 1.37 |
| Large | 400 | 400 | 10 | 10 | 45.57 | 8.92 | 0.76 |

The timetable quality is measured based on the number of the soft constraint violations (penalty cost). Each violation of a soft constraint will be penalized '1' for each student who is involved in this situation (Mcmullan, 2007). All hard constraints must be satisfied since we only deal with feasible solutions, which is usually the case for the majority of research in this domain.

## Methodology

In this study, non-parametric Acceptance Criterion (AC) algorithm is propose, which motivate to propose an Adaptive Acceptance Criterion (AAC) algorithm for the optimization problems. AC and AAC starts with generating an initial solution (or initial clusters partition) and iteratively explores its neighbor solutions (other solution), looking for a better one by any algorithm. The neighbor solution is accomplished by restructure the current solution (or partition) using some neighborhood structures. The initial solution for the medical clustering problem is presented in section 3.1 and the neighborhood structures are described in 3.1.1. Where, the initial solution for the university course timetabling problem is presented in section 3.2 and the neighborhood structures are described in 3.2.1.

### *Initial Solution for Medical Clustering Problem*

In this study, *Multi K-Means*algorithm (Holland, 1975) is used as in (Kittaneh *et al*., 2012; Abuhamdah, 2012) to generate the initial solution partition for medical clustering problem. *Multi K-Means* algorithm structure is similar to *K-Means*algorithm structure (which is well known by its simplicity to deal with a huge amount of data patterns). *K-Means* aims to minimize the squared error established from Euclidean distance (Equation (1)), where the algorithm takes *X* as input parameters and partitions the set of *n* objects into *K* clusters. The basic difference between them is that, in *K-Means* a random cluster centers (centroids) is defined, whilst in *Multi K-Means*, we initially define a random cluster centers (centroids), then the final cluster centers is determined after the *K-Means* is restarted for 50 times as recommended (Holland, 1975) by re-computing the centroids *vj* of cluster *j* as illustrated in Equation 2:

$$v_j = \frac{1}{C_i}\sum_{i=1}^{c_i} x_{ij} \quad (2)$$

### *Neighborhood Structures for Medical Clustering Problem*

Two neighborhood structures (i.e., N1and N2) are used (which have been widely used in the literature) as in (Kittaneh *et al*., 2012; Abuhamdah, 2012). The neighborhood structures are:

- N1: Randomly select one pattern from each cluster to swap their data with other pattern in other clusters.
- N2: Randomly select two different patterns from the same cluster and swap their data.

### *Initial Solution for University Course Timetabling Problem*

In this study, the initial solution generated by a constructive heuristic that was proposed in (Mcmullan, 2007) for the course timetabling problems. There are three phases in the constructive heuristic as follow: largest degree heuristic (Landa-Silva and Obit, 2008), neighborhood search and tabu search. The constructive

heuristic starts with an empty timetable and consecutively invokes three phases to generate a feasible timetable.

In the first phase, all unscheduled courses are sorted depend on the number of students conflict with other courses. Then, the student who has the highest number of conflicts compared to the other selected courses is selected first. The selected course may assign to any random feasible timeslot-room. However, if cannot find a feasible room for this course, it will be assigned to any room. If all the courses have been scheduled to feasible timeslot-rooms, we ignore phases 2 and 3. Otherwise, phases 2 and 3 are invoked to achieve the feasibility.

Phase 2, employs a simple decent algorithm to reduce the hard constraint violations. The neighborhood solution is generated by either moving one course from its current timeslot-room into another random timeslot-room, or it randomly selects two courses and swaps their rooms and timeslots. In both cases, the new solution is accepted if the move does not violate any hard constraints and the quality of the generated timetable is better than the previous solution quality in terms of hard constraints violation. Phase 2 is terminated after ten non-improving iterations. If the solution is feasible, we ignore phase 3, otherwise, phase 3 is invoked.

Phase 3, employs a tabu search algorithm that explores neighboring solutions in a similar way to phase 2, but it also maintains a tabu list to prevent certain moves being made for a certain number of iterations. The size of the tabu list is calculated by $tl = rand(10)+\delta * nc$, where $rand(10)$ is a random number between 0 and 10, $nc$ is the number of events that violate the hard constraints and $\delta$ is a constant which is set to 0.6 (Mcmullan, 2007). This phase will stop after 1000 non-improving iterations. If the generated solution is infeasible, re-call the constructive heuristic to generate a new solution from scratch until a feasible solution is found.

### *Neighborhood Structures for University Course Timetabling Problem*

Two neighborhood structures (i.e., NS1and NS2) are used (which have been widely used in the literature) as in (Abuhamdah and Ayob, 2009). The neighborhood structures are:

- NS1: Randomly select two courses and swap their timeslots (and rooms if feasible)
- NS2: Randomly select a course, feasible timeslot and feasible room and move the course to the new timeslot (and move the course to the new room if necessary)

## AAC Algorithm

This work is motivated by the strength of local based algorithms in exploiting the search space (intensification process) (Ayvaz *et al*., 2012). However, the disadvantage of the local search approaches (Ayvaz *et al*., 2012) and the descent heuristic techniques (Schaerf, 1999) is in tuning the parameters and incapable of escaping from local optima. Therefore, in this study, a non-parametric Acceptance Criterion (AC) algorithm is proposed to intensify the search and overcome the limitation of the parameter tuning, which motivates to propose an Adaptive Acceptance Criterion (AAC) algorithm to overcome the other limitation by increase the search exploration (or the diversification mechanism). Note that, the discussion bellow is for minimization on the clustering problem domain, which is similar to the minimization process of the university course timetabling problem domain (except in the initial solution, neighborhood structures and the terms).

AC is a simple mechanism based on stored value that may able to control the diversification with good quality results and produce a consistent result for different problems. AC starts with a given *Multi K-Means* partitions i.e., the initial solution ($S_{initial}$) is generated by *Multi K-Means* algorithm and then iteratively improve the constructed solution by generating a neighbor solution (candidate solution) by using neighborhood structure(s). AC always accept the generated candidate (new) solution ($S_{working}$) if the quality is better (less) than the best solution ($S_{Arrange}$) value, or can probably accept a little worse solution by an adaptive acceptance criterion (AC objective function). AC Acceptance Criterion (AC) as illustrated in Equation 3, adaptively accepts the worse solution if $AC$ value is greater than or equal to the stored value (*SV, which is stored based or the best and candidate solutions*), otherwise $S_{working}$ will be rejected. This process will be repeated until the stopping condition is met:

$$AC = (S_{Arrange} / S_{working}) \tag{3}$$

$SV$ value is initially stored when a new best solution found using $AC$ acceptance criterion (see Equation (3)) and later when there is any new best solution found then, $SV$ will be updated with the lowest value using $AC$ acceptance criterion (which control the diversification with a little worse solution). For example, we initially calculate $SV$ by the first improved value using $AC$ criterion. Later on, when there is any enhancement for best solution found $S_{Arrange}$, we re-compute $AC$ for the next iterations, however, if the computed $AC$ is lowest than the $SV$ value, then $SV$ will be updated with $AC$ value and so on. The process of updating $SV$ is governed to increase a little diversification based on the solution improvement. In other words, when $SV$ value is smaller or equal to $AC$ value, then the candidate solution accepted for the next iteration. Please note that, in case of accepted solutions by the best solution found, then in Equation 3, we need to switch between solutions (i.e., $AC= S_{working} / S_{Arrange}$ ) as the $S_{Arrange}$ is lowest than the new best solution $S_{working}$. However, initialize SV value

can be done in two ways, the first based on the first improvement as discussed and the second, we can maximize the problem for some iteration to identify *SV*.

Figure 1 show the pseudo code for the AC approach, where the lists of notations that are used in Fig. 1 (AC algorithm) is for clustering problem. Table 3 differentiate between the notation used in the clustering problem domain and the notation used in university course timetabling problem domain.

Figure 1 shows that, AC starts by initializing *SV* equal to zero and the only required parameter setting, the stopping condition ($N_{iterations}$), where the initial solution is generated using *Multi K-Means* ($S_{initial}$) as in Step-1.
In the improvement phase (Step-2), we generate some candidate solutions (in this case, five candidate solutions are generated) as in (Kittaneh *et al*., 2012; Abuhamdah, 2012) for each neighborhood structure (i.e., N1 and N2) and the best candidate solution is selected as the candidate solution ($S_{working}$) as in Step-2.1. Later on, there are two cases to evaluate the candidate solution as follows:

### *Good Solution*

If $f(S_{working})$ is better than $f(S_{Arrange})$, then we update *SV* value by the computing *AC* (i.e. $AC=S_{working} / S_{Arrange}$) value in case of *SV* equal to zero or in case of *SV* is greater than *AC,* otherwise *SV* value not updated. After, $S_{working}$ is accepted as a current solution (i.e. $S_{source} \leftarrow S_{working}$) and the best solution is updated (i.e. $S_{Arrange} \leftarrow S_{working}$) as in Step-2.2. Note that, we have switch the solutions in Equation 3 as $S_{working}$ is a better solution than $S_{Arrange.}$

### *Little Worse or Bad Solution*

The quality between $S_{working}$ and $S_{Arrange}$ is compared by computing *AC* value as in Equation 3, (i.e., $AC= S_{Arrange} / S_{working}$ )) in case if *SV* initialized with a value (not equal to zero). If *AC* is greater than or equal to *SV* then $S_{working}$ is accepted and the current solution is updated (i.e. $S_{source} \leftarrow S_{working}$). Otherwise, $S_{working}$ will be rejected.
Otherwise, the process of the step 2 continues until the stopping condition is met (i.e. $Iterations > N_{iterations}$). However, if the stopping condition is met, then the termination phase (Step-3) is active to return the best solution found $S_{Arrange}$ (or the best minimal distance).

However, the preliminary experiments on AC algorithm idea show that, AC algorithm can easily trapped in local optima when we are dealing with small datasets with zero optimal solution. For example, when $S_{Arrange}$ is equal to 2 and $S_{working}$ is equal to 1 in case of better solution, then *AC* value is 0.5 and *SV* is 0.5, where in the next iteration the candidate solution will not accepted as a little worse solution if it is greater than 2 (as the new $S_{Arrange}$ is 1 and SV is 0.5), but sometimes more worse solution can improve more, which motivates to increase the diversification strategy in AC. There are many strategy used for diversification, however, the Adaptive Randomized Descent Algorithm (ARDA) which was proposed by Abuhamdah and Ayob (2010) employ an adaptive mechanism proposed for their acceptance criterion in similar optimization (or minimization) problem which allow some slightly worse solution to be accepted and helps to escape from a local optima. The idea in ARDA motivates to utilize it in *AC* and termed as *AAC* algorithm.

ARDA mechanism can adaptively attempt to escape from local optima by intelligently updating the threshold value when the search traps in local optima. This is done by estimating an appropriate threshold value based on the search history. ARDA mechanism based on array ($L_{EV}$) of estimated values *EV* with their frequencies. These estimated values stored based on the difference between the new and old solutions found (i.e., $EV= f(S_{working}) - f(S_{Arrange})$), where when a new *EV* value found and it is different with the other *EV* values in the array, is then the array updated by adding the new *EV* value with frequency (or the number of repetition) is equal to one. In each improvement on the best solution, the new *EV* value is added and if the *EV* value is already in the array, then we add one to the frequency of *EV* value. However, in each time we update the array, then we rearrange the array in descending order based on their frequency value, in which the first value in the array is the value with the highest frequency value. ARDA mechanism starts when a counter ($C_{idle\text{-}iterations}$) of the idle improvement rate (*idle-iterations*) is met, then update the threshold value as their acceptance criterion by adding the value of the highest frequency value to their threshold value. However, AC algorithm threshold value is based on the solutions quality, therefore, we use ARDA mechanism to add the value for the best solution found $f(S_{Arrange})$ in *AC* acceptance criterion and termed as *AAC* algorithm. For example, when $S_{Arrange}$ is equal to 2 and $S_{working}$ is equal to 1 in case of better solution, then *AC* value is 0.5 and *SV* is 0.5, where in the next iteration the candidate solution will not accept the $S_{working}$ solution if it is greater than 2 (as the new $S_{Arrange}$ is 1 and SV is 0.5). However, if there the idle of improvement rates *idle-iterations* is met, then ARDA mechanism works to add *EV* value (e.g. EV is equal to 2) of the highest frequency to the $S_{Arrange}$ solution for increase the diversification in case *AC* calculation for the worse solution, in which *AC* value will be equal to 1.5 (i.e., AC= (1+2) / 2 )). Note that, the case discussed above is in case of the near optimal solution, where if it is not near optimal, so adding *EV* will increase little diversification. Figure 2 shows the pseudo code for the extensions of AC approach (AAC), where the combination between Fig. 1 and 2 and eliminating the duplication illustrate AAC algorithm.

As in Fig.2, AAC starts by initializing the idle improvement iterations (*idle-iterations),* Estimated Value (EV) and the array ($L_{EV}$) of the estimated values equal to zero.

Table 3. The notations for clustering and university course timetabling datasets

| Notation in Clustering | Notation in Course timetabling | Description |
|---|---|---|
| $S_{initial}$ | $S_{initial}$ | initial solution |
| $f(S_{initial})$ | $f(S_{initial})$ | quality of initial solution |
| $S_{Arrange}$ | $S_{best}$ | best solution found |
| $f(S_{Arrange})$ | $f(S_{best})$ | the quality of best solution found |
| $S_{source}$ | $S_o$ | the current solution |
| $f(S_{source})$ | $f(S_o)$ | the quality of the current solution |
| $S_{working}$ | $S^*$ | the candidate solution |
| $f(S_{working})$ | $f(S^*)$ | the quality of the candidate solution |
| $N_{iterations}$ | $N_{iterations}$ | number of iterations |
| *Iterations* | *Iterations* | iteration counter |
| *AC* | *AC* | acceptance criterion (objective function) |
| *SV* | *SV* | Stored value |
| $L_{EV}$ | $L_{EV}$ | array of the estimated values |
| *idle-iterations* | *idle-iterations* | number of the idle iterations |
| $C_{idle\text{-}iterations}$ | $C_{idle\text{-}iterations}$ | counter for the idle iterations |
| *EV* | *EV* | estimated value |

**Procedure** Acceptance Criterion Algorithm (AC)

**Step-1:** Initialization Phase
*Determine K-mean solution (initial solution)* $S_{initial}$ *and* $f(S_{initial})$;
$S_{Arrange} = S_{initial}$; $f(S_{Arrange}) = f(S_{initial})$;
$S_{source} = S_{initial}$; $f(S_{source}) = f(S_{initial})$;
*Set the iterations number* $N_{iterations}$;
*Set SV and Iterations equal to zero ;*

**Step-2:** Improvement (Iterative) Phase
**repeat** (while termination condition is not satisfied)
*Add Iterations by one;*
**Step-2.1:** Selecting candidate solution $S_{working}$
*Generate candidate solutions by applying all neighborhood structures (N1 and N2) and the best solution between them consider as candidate solution* ($S_{working}$);
**Step-2.2:** Accepting Solution
**if** $f(S_{working}) < f(S_{Arranget})$
*Calculate AC; note that we switch the solutions "* $AC = (S_{working} / S_{Arrange})$ *"*
*if SV ==0 (initialization)*
$SV = AC$;
*end if*
*if SV > AC*
$SV = AC$;
*end if*
$S_{Arrange} = S_{working}$; $f(S_{Arrange}) = f(S_{working})$;
$S_{source} = S_{working}$; $f(S_{source}) = f(S_{working})$;
else
*if SV <> 0*
*if AC >= SV then*
$S_{source} = S_{working}$;
*end if*
*end if*
**end if**
**until** *Iterations* > $N_{iterations}$ (termination condition is met)

**Step-3:** Termination phase
*Return the best solution found* $S_{Arrange}$

Fig. 1. Pseudo code for Acceptance Criterion Algorithm (AC) for optimization problems

**Procedure** Adaptive Acceptance Criterion Algorithm (AAC)

**Step-1:** Initialization Phase
*Set EV,* $C_{idle\text{-}iterations}$, *equal to zero ;*
*Set* $L_{EV}$ *equal to zero value with zero frequency;*
*Set idle-iterations;*

**Step-2:** Improvement (Iterative) Phase
**repeat** (while termination condition is not satisfied)
*Add Iterations by one;*
**Step-2.1:** Selecting candidate solution $S_{working}$

**Step-2.2:** Accepting Solution
**if** $f(S_{working}) < f(S_{Arranget})$
$C_{idle\text{-}iterations} = 0$;
$EV = f(S_{Arrange}) - f(S_{working})$;
*Update the frequency of EV in the* $L_{EV}$ *(or add EV in* $L_{EV}$ *if it is not exist);*
*Sort the EV elements in* $L_{EV}$ *in descending order based on their frequency or if equivalence then Ascending based on their value;*
$S_{Arrange} = S_{working}$; $f(S_{Arrange}) = f(S_{working})$;
$S_{source} = S_{working}$; $f(S_{source}) = f(S_{working})$;
else
$C_{idle\text{-}iterations} = C_{idle\text{-}iterations} + 1$;
*end if*
*end if*
**end if**

**Step 2.3:** Idle Iteration
*if* $C_{idle\text{-}iterations} >=$ *idle-iterations*
$EV = L_{EV}[0]$; *// get the first element in A*
*Rotate left all elements in* $L_{EV}$;
*Calculate AC by adding EV to* $f(S_{Arrange})$;
*if AC >= SV then*
$S_{source} = S_{working}$;
$C_{idle\text{-}iterations} = 0$
*end if*
**until** *Iterations* > $N_{iterations}$ (termination condition is met)

**Step-3:** Termination phase
*Return the best solution found* $S_{Arrange}$

Fig. 2. Pseudo code for the Adaptive Acceptance Criterion Algorithm (AAC) that extend Fig. 1 to the optimization problems

In the improvement phase (Step-2), in case of the good solution accepted, set the counter for the idle iterations $C_{\text{idle-iterations}}$ equal to zero, we calculate the estimated value (i.e. $EV= f(S_{Arrange})-f(S_{working})$), updates the frequency of $EV$ value in $L_{EV}$ or we added $EV$ to $L_{EV}$ if it does not exist, rearrange the array $L_{EV}$ values in descending order based on their frequency or if the values are equivalence then rearrange the equivalence values in ascending order, then $S_{working}$ is accepted as a current solution (i.e. $S_{source} \leftarrow S_{working}$) and the best solution is updated (i.e., $S_{Arrange} \leftarrow S_{working}$) as in Step-2.2. While, in case of little worse or bad solution accepted, nothing changed except if the worse solution rejected, then the counter for the idle iterations ($C_{idle\text{-}iterations}$) is added by one. Finally, in the additional idle iterations phase (Step-3), the counter for the idle iterations ($C_{idle\text{-}iterations}$) is compared to the maximum number of idle iterations (*idle-iterations*) and if it is greater or equal (i.e., $C_{idle\text{-}iterations} >= idle\text{-}iterations$), then $EV$ is generated by the first value of $L_{EV}$ (as it has the highest frequency value), rotate left all elements in $L_{EV}$ to use the next value in the next idle phase in case of non-improvement, then recomputed $AC$ by adding $EV$ to the best solution found (i.e. $AC=S_{working} / (S_{Arrange} + EV)$) and if accepted set the counter of the idle iterations to zero. Otherwise, the process continues as it is illustrated in Fig. 1.

However, the limitation of AAC algorithm, that we added a new parameter (i.e. idle iterations, which is equal to 10 as in (Abuhamdah and Ayob, 2010), which need investigation for each problem, but it consider acceptable as it is the only parameter for the proposed AAC.

## Results

In this study, AC and AAC algorithms are run 20 times as in (Kittaneh *et al*., 2012; Abuhamdah, 2012) for, medical clustering problem by using 6 datasets that are available in the UCI machine learning repository (http://archive.ics.uci.edu/ml/index.html). Also, AC and AAC algorithms are run 11 times as in (Abuhamdah and Ayob, 2009) for, university course timetabling problem by using 11 datasets (Socha benchmark datasets). In both problem domains, the algorithms run on a PC with an Intel dual core 1.8 MHz, 2 GB RAM and were programmed for clustering using Java language as in (Kittaneh *et al*., 2012; Abuhamdah, 2012) and for course timetabling using Matlab as in (Abuhamdah and Ayob, 2009). There is only one parameter is used in the AC algorithms for the stopping conditions ($N_{\text{iterations}}$) which is equal to 100,000 iterations for clustering as in SA with prolonging the search and termed as IISA (Kittaneh *et al*., 2012) and GD (Abuhamdah, 2012), where for course timetabling is equal to 200,000 iterations as in SA and GD (Abuhamdah and Ayob, 2009). In addition, there is another parameter is used in AAC for the number of idle iterations (idle-iterations) which is equal to 600 for clustering as in (Abuhamdah, 2012) and 10 for timetabling as in (Abuhamdah and Ayob, 2010). Table 4 shows the quality of the initial solution (or clusters partitions) obtained by Multi K-Means algorithm for each dataset using two calculations ways of the 6 datasets. Where, Table 5 shows the quality of the initial solution (or timetable) obtained by constructive heuristics (Mcmullan, 2007; Landa-Silva and Obit, 2008) for each dataset of the 11 datasets.

In order to investigate the performance of AC and AAC algorithms, Table 6 shows the comparison between AC and AAC algorithms using the between objects calculation. Tables 6-14, illustrate the best minimal distance quality (*fmin*), the average score (*favg*) and the standard deviation ($\sigma$) for the 20 runs. In each table, the best results (*fmin*) are presented in bold.

Results in Table 6 indicates that, AAC algorithm is able to produce good quality solution outperformed AC algorithm solutions in all datasets referring to the best minimal distance *fmin*, the average score *favg* and the standard deviation $\sigma$ (except in *B.L.D*, *P.I.D* and *L.C* datasets, AC algorithm obtained better standard deviation than AAC algorithm, in addition to the average score in *L.C* dataset). Table 7 shows the comparison between AAC, IISA and GD algorithms using the between objects calculation for the six medical clustering datasets.

Table 7 shows that, AAC algorithm also is outperformed IISA and GD algorithms in all datasets referring to the best minimal distance *fmin*, the average score *favg* and the standard deviation $\sigma$ (where the standard deviation for GD is not known). Table 8 shows the comparison between AAC approach and other local hybrid meta-heuristic searches in the literature using the between objects calculation.

According to Table 8, AAC algorithm is able to produce high quality solution outperformed IISA (Kittaneh *et al*., 2012), MGD (Abuhamdah, 2012), ISA-MGD (Abuhamdah *et al*., 2012) and AGD (Abuhamdah *et al*., 2014) algorithms in all datasets referring to the *fmin* and *favg* (except in *B.C* dataset, MGD algorithm obtained same result with AAC algorithm). Figure 3 shows a 3D scatter graph for *Multi K-Means*, AC and AAC algorithms over *H.S* dataset using between objects calculation. *H.S* dataset has two clusters represented by two colors (red and green).

### *Partitions in between Objects Calculation*

Figure 3a show that, the initial minimal distance obtained by Multi K-Means is 2463.972. Where in Fig. 3b, the best minimal distance obtained by AAC algorithm is equal to 947.64. However, in Fig. 3; the best minimal distance obtained by AAC algorithm (i.e., 947.64) is slightly different from AC algorithm (i.e., 987.68) to show their differences, therefore AC algorithm graph not included.

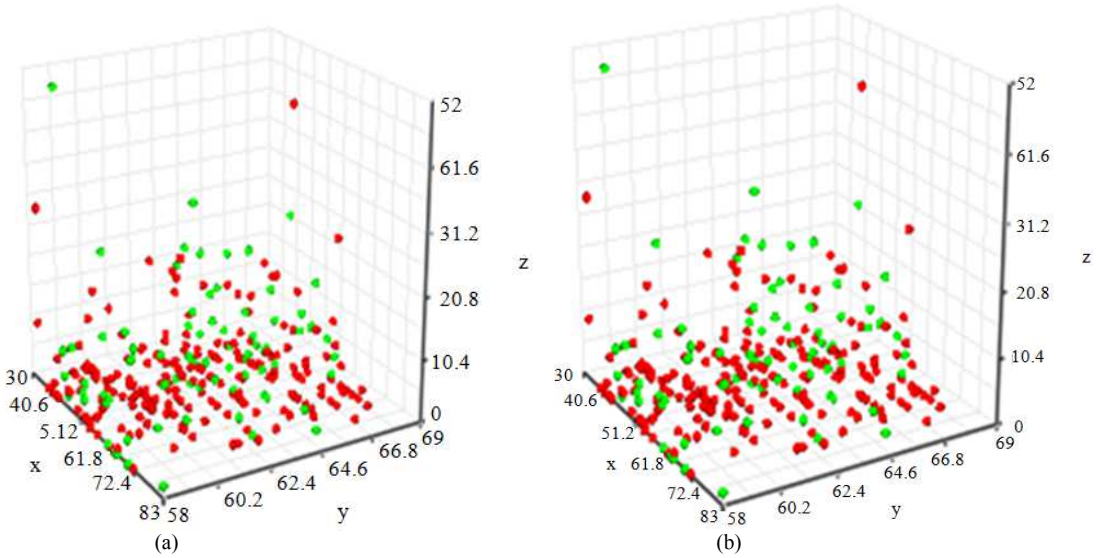


Fig. 3. Scatter graph for Multi K-Means and AAC algorithm over H.S dataset using between objects calculation. Partitions in between Objects Calculation (a) The Initial Minimal Distance Found by Multi K-Means is 2463.972 (b) The Best Minimal Distance by AAC is 947.64

Table 4. Initial solution quality for each dataset of the medical clustering using two calculations

| Datasets | Initial solution quality for the between objects | Initial solution quality for the between centres |
|---|---|---|
| *H.S* | 2463.97 | 3626.530 |
| *B.L.D* | 17258.71 | 22646.890 |
| *P.I.D* | 100880.39 | 102398.583 |
| *B.C* | 6379.69 | 5360.710 |
| *T.D* | 3178.71 | 2459.620 |
| *L.C* | 182.57 | 168.520 |

Table 5. Initial solution quality for each dataset of the university course timetabling

| Datasets | Initial solution quality |
|---|---|
| Small 1 | 301 |
| Small 2 | 332 |
| Small 3 | 463 |
| Small 4 | 342 |
| Small 5 | 312 |
| Medium 1 | 775 |
| Medium 2 | 834 |
| Medium 3 | 894 |
| Medium 4 | 953 |
| Medium 5 | 845 |
| large | 1974 |

Moreover, the (AC and AAC) algorithms investigated using the between centers calculation for the six medical clustering datasets as in Tables 9, 10 and 11. Table 9 shows the comparison between AC and AAC algorithms; Table 10 shows the comparison between AAC, IISA and GD algorithms; and Table 11 shows the comparison between AAC approach and other local hybrid meta-heuristic searches in the literature.

Tables 9, 10 and 11 shows the comparison using the calculation of between centers, where the observation (Table 9) indicates that, AAC algorithm is outperformed AC algorithm in all datasets referring to the *fmin* and *favg* (except in *L.C* dataset they obtained same *fmin* and the *favg* for *B.L.D* and *L.C* datasets in AC algorithm is better than AAC algorithm), where $\sigma$ in AC is better than AAC all datasets *(*except in *B.L.D* dataset). In addition, the observation (in Table 10) shows that, AAC algorithm also is outperformed IISA and GD algorithms in all datasets referring to the *fmin*, *favg* and $\sigma$ (where in *L.C* dataset MGD is obtain same *fmin* in addition that in *H.S* and *T.D* datasets the $\sigma$ for IISA is better than AAC and the $\sigma$ for GD is not known). Where the comparison in Table 11 with the other approaches in the literature shows that, AAC algorithm is able to produce high quality solution outperformed IISA (Kittaneh *et al*., 2012), MGD (Abuhamdah, 2012) and ISA-MGD (Abuhamdah *et al*., 2012) and AGD (Abuhamdah *et al*., 2014) algorithms in all datasets referring to the *fmin* and *favg* (except in *L.C* dataset, MGD and AGD algorithms obtained same *fmin*).

### *Partitions in between Centers Calculation*

Figure 4 shows a 3D scatter graph for *Multi K-Means*, AC and AAC algorithms on over *H.S* dataset using for the between centers calculation.

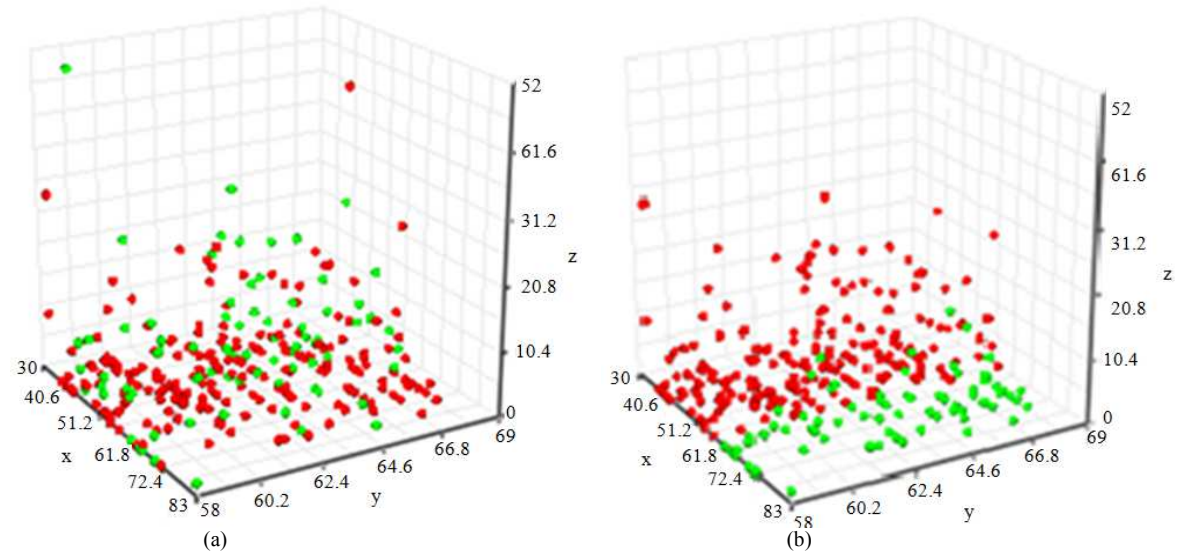


Fig. 4. Scatter graph for Multi K-Means and AAC algorithm over H.S dataset. Partitions in between centers calculation (a) The Initial Minimal Distance Distance Found by Multi K-Means (b) The Best Minimal by AAC is 2703.11

Table 6. Results obtained by between objects calculation for AC and AAC algorithms out of 20 runs on six datasets

| | *fmin* | | favg | | Std. Dev. (σ) | |
|---|---|---|---|---|---|---|
| Dataset | AC | AAC | AC | AAC | AC | AAC |
| H.S | 987.68 | 947.64 | 1020.94 | 969.61 | 24.23 | 14.33 |
| B.L.D | 5403.57 | 5352.12 | 5435.70 | 5429.16 | 25.92 | 34.08 |
| P.I.D | 21434.00 | 21000.35 | 21707.73 | 21573.30 | 238.59 | 329.77 |
| B.C | 1951.00 | 1937.00 | 1982.85 | 1949.00 | 36.40 | 13.01 |
| T.D | 902.89 | 844.74 | 918.89 | 866.51 | 15.78 | 13.42 |
| L.C | 157.80 | 157.62 | 158.68 | 158.90 | 0.59 | 0.67 |

Table 7. Results obtained by between objects calculation for AAC, IISA (Kittaneh *et al*., 2012) and GD (Abuhamdah, 2012) algorithms out of 20 runs on six datasets

| | *fmin* | | | favg | | | Std. Dev. (σ) | | |
|---|---|---|---|---|---|---|---|---|---|
| Dataset | AAC | IISA | GD | AAC | IISA | GD | AAC | IISA | GD |
| H.S | 947.64 | 987.77 | 1215.22 | 969.61 | 1049.03 | 1311.76 | 14.33 | 32.65 | - |
| B.L.D | 5352.12 | 5771.59 | 6476.68 | 5429.16 | 6028.67 | 6745.21 | 34.08 | 140.29 | - |
| P.I.D | 21000.35 | 24920.39 | 29311.15 | 21573.30 | 26038.86 | 32311.91 | 329.77 | 796.96 | - |
| B.C | 1937.00 | 2338.00 | 2085.00 | 1949.00 | 2430.05 | 2192.70 | 13.01 | 52.17 | - |
| T.D | 844.74 | 1228.80 | 973.44 | 866.51 | 1375.23 | 1055.68 | 13.42 | 70.08 | - |
| L.C | 157.62 | 158.98 | 159.27 | 158.90 | 161.14 | 161.38 | 0.67 | 0.86 | - |

Table 8. Comparison between AAC algorithm and other approaches in the literature using between objects calculation

| | 20 Runs-Minimal Distance Calculated as between Objects | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | AAC | | IISA | | MGD | | ISA-MGD | | AGD | |
| Dataset | *fmin* | *favg* | *fmin* | *favg* | *fmin* | *favg* | *fmin* | *favg* | *fmin* | *favg* |
| *H.S* | 947.64 | 969.61 | 987.77 | 1049.03 | 1023.96 | 1069.57 | 1014.05 | 1043.21 | 1016.00 | 1044.53 |
| *B.L.D* | 5352.12 | 5429.16 | 5771.59 | 6028.67 | 5509.21 | 5809.55 | 5466.33 | 5631.02 | 5483.19 | 5599.16 |
| *P.I.D* | 21000.35 | 21573.30 | 24920.39 | 26038.86 | 23281.12 | 24239.71 | 22919.27 | 24118.18 | 23004.09 | 23897.48 |
| *B.C* | 1937.00 | 1949.00 | 23380.00 | 2430.05 | 1937.00 | 2088.40 | 2090.61 | 2124.76 | 1937.00 | 1964.80 |
| *T.D* | 844.74 | 866.51 | 1228.80 | 1375.23 | 893.33 | 946.69 | 892.57 | 929.28 | 898.97 | 919.92 |
| *L.C* | 157.62 | 158.90 | 158.98 | 161.14 | 159.27 | 161.38 | 157.80 | 159.20 | 158.98 | 159.98 |

Table 9. Results obtained by between centers calculation for AC and AAC algorithms out of 20 runs on six datasets

| | *fmin* | | favg | | Std. Dev. (σ) | |
|---|---|---|---|---|---|---|
| Dataset | AC | AAC | AC | AAC | AC | AAC |
| H.S | 2721.36 | 2703.11 | 2721.7700 | 2719.88 | 0.5700000 | 5.76000 |
| B.L.D | 10498.90 | 10493.95 | 10708.6700 | 10722.59 | 296.8700000 | 293.31000 |
| P.I.D | 48909.20 | 48851.62 | 54266.2900 | 54029.54 | 4830.1100000 | 4856.38000 |
| B.C | 3007.32 | 2778.00 | 3029.1655 | 3012.35 | 31.5232438 | 60.92018 |
| T.D | 2039.89 | 2028.27 | 2043.0300 | 2032.80 | 5.6400000 | 8.38000 |
| L.C | 151.62 | 151.62 | 152.7400 | 152.89 | 0.9900000 | 1.09000 |

Table 10. Results obtained by between centers calculation for AAC, IISA (Kittaneh *et al*., 2012) and GD (Abuhamdah, 2012) algorithms out of 20 runs on six datasets

| | *fmin* | | | favg | | | Std. Dev. (σ) | | |
|---|---|---|---|---|---|---|---|---|---|
| Dataset | AAC | IISA | GD | AAC | IISA | GD | AAC | IISA | GD |
| H.S | 2703.11 | 2721.36 | 2721.36 | 2719.88 | 2722.00 | 2726.37 | 5.76000 | 0.60 | - |
| B.L.D | 10493.95 | 10498.90 | 10498.90 | 10722.59 | 10855.71 | 10839.38 | 293.31000 | 354.91 | - |
| P.I.D | 48851.62 | 48909.20 | 48909.20 | 54029.54 | 54751.22 | 56357.49 | 4856.38000 | 4889.73 | - |
| B.C | 2778.00 | 2778.00 | 3014.72 | 3012.35 | 3104.40 | 3326.92 | 60.92018 | 304.30 | - |
| T.D | 2028.27 | 2039.89 | 2039.89 | 2032.80 | 2054.09 | 2051.72 | 8.38000 | 4.85 | - |
| L.C | 151.62 | 152.37 | 151.62 | 152.89 | 154.28 | 153.70 | 1.09000 | 1.30 | - |

Table 11. Comparison between AAC algorithm and other approaches in the literature between centers calculation

| | 20 Runs – Minimal Distance Calculated as between Centers | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | AAC | | IISA | | MGD | | ISA-MGD | | AGD | |
| Dataset | *fmin* | *favg* | *fmin* | *favg* | *fmin* | *favg* | *fmin* | *favg* | *fmin* | *favg* |
| *H.S* | 2703.11 | 2719.88 | 2721.36 | 2722.00 | 2721.36 | 2721.594 | 2721.36 | 2730.16 | 2721.36 | 2721.594 |
| *B.L.D* | 10493.95 | 10722.59 | 10498.90 | 10855.71 | 10498.90 | 10836.970 | 10498.90 | 10969.26 | 10498.90 | 10749.850 |
| *P.I.D* | 48851.62 | 54029.54 | 48909.20 | 54751.22 | 48909.20 | 56159.740 | 48909.20 | 57098.70 | 48909.20 | 51248.780 |
| *B.C* | 2778.00 | 3012.35 | 2778.00 | 3104.40 | 3014.72 | 3316.570 | 3007.32 | 3281.79 | 3010.70 | 3227.381 |
| *T.D* | 2028.27 | 2032.80 | 2039.89 | 2054.09 | 2039.89 | 2047.040 | 2070.16 | 2079.52 | 2039.89 | 2046.090 |
| *L.C* | 151.62 | 152.89 | 152.37 | 154.28 | 151.62 | 153.490 | 152.37 | 153.04 | 151.62 | 153.230 |

Figure 4a shows, the initial minimal distance obtained by Multi K-Means is 2463.972. Where in Fig. 4b, the best minimal distance obtained by AAC algorithm is equal to 947.64. Also, in Fig. 4; the best minimal distance obtained by AAC algorithm (i.e., 2703.11) is slightly different from AC algorithm (i.e., 2721.36) to show their differences, therefore AC algorithm graph not included.

Furthermore, the (AC and AAC) algorithms investigated using the eleven university course timetabling problems as in Tables 12 to 14. Table 12 shows the comparison between AC and AAC algorithms; Table 13 shows the comparison between AAC, SA and GD algorithms; and Table 14 shows the comparison between AAC approach and other local hybrid meta-heuristic searches in the literature.

According to the results in Table 12, AAC algorithm is able to produce good results outperformed AC algorithm in all Medium and large datasets, where they obtain same best result in all the small datasets, in addition that AAC standard deviation for all the small and medium 2 datasets is better than AC algorithm. The results in Table 12 also show that, AAC average scores are better than AC. Where the comparison in Table 13 indicates that, AAC outperformed SA and GD algorithms in all datasets; except in small 5 dataset they obtain the same best result, in addition that GD obtains same best result with AAC for small 1, 2 and 4 datasets. Table 13 also shows that SA and AAC obtained same $\sigma$, SA $\sigma$ is better than AAC in Medium 5 dataset and GD $\sigma$ is better than AAC in small 4 dataset.

Table 14 shows that AAC algorithm is able to produce good quality solution are equivalence with some approaches (i.e., A1, A2…, etc) in the small datasets and comparable with other approaches (i.e., A1, A2,.., A26) in the literature in the other datasets. The best results for Medium 1, Medium 2 and Medium 3 datasets is obtained by Abuhamdah *et al*. (2013), and the best results for Medium 4 is obtained by Turabieh and Abdullah (2009), whilst, the best results for Medium 5 and large datasets is obtained by Turabieh *et al*. (2010).

However, all these result are obtained with population based algorithms which makes the percentage deviation ($\Delta$ (%)) of AAC algorithm between 0.56 and 1.16 is acceptable.

Table 12. Results for AC and AAC algorithms using the eleven datasets for course out of 11 runs on eleven datasets

| | *fmin* | | favg | | Std. Dev. (σ) | |
|---|---|---|---|---|---|---|
| Dataset | AC | AAC | AC | AAC | AC | AAC |
| Small 1 | 0 | 0 | 1.09 | 0.63 | 1.22 | 0.81 |
| Small 2 | 0 | 0 | 2.09 | 0.73 | 1.51 | 1.01 |
| Small 3 | 0 | 0 | 1.45 | 1.18 | 1.21 | 1.17 |
| Small 4 | 0 | 0 | 1.18 | 1.09 | 0.75 | 0.70 |
| Small 5 | 0 | 0 | 1.00 | 0.64 | 1.18 | 0.81 |
| Medium 1 | 122 | 75 | 134.36 | 91.82 | 10.03 | 10.73 |
| Medium 2 | 114 | 61 | 124.90 | 70.18 | 8.77 | 6.43 |
| Medium 3 | 157 | 124 | 163.73 | 137.36 | 6.63 | 9.59 |
| Medium 4 | 108 | 69 | 112.91 | 74.27 | 4.78 | 5.16 |
| Medium 5 | 116 | 96 | 129.55 | 117.45 | 11.18 | 12.32 |
| large | 724 | 634 | 740.82 | 660.27 | 13.56 | 20.86 |

Table 13. Results for AAC, SA (Abuhamdah and Ayob, 2009) and GD (Abuhamdah, 2012) algorithms using the eleven datasets for course out of 11 runs on eleven datasets

| | *fmin* | | | favg | | | Std. Dev. (σ) | | |
|---|---|---|---|---|---|---|---|---|---|
| Dataset | AAC | SA | GD | AAC | SA | GD | AAC | SA | GD |
| Small 1 | 0 | 1 | 0 | 0.63 | 2.72 | 2.45 | 0.81 | 1.01 | 1.630 |
| Small 2 | 0 | 1 | 0 | 0.73 | 2.72 | 1.72 | 1.01 | 1.01 | 1.500 |
| Small 3 | 0 | 1 | 1 | 1.18 | 2.54 | 1.69 | 1.17 | 1.29 | 1.580 |
| Small 4 | 0 | 1 | 0 | 1.09 | 2.09 | 2.45 | 0.70 | 0.94 | 1.630 |
| Small 5 | 0 | 0 | 0 | 0.64 | 2.63 | 1.72 | 0.81 | 1.63 | 1.790 |
| Medium 1 | 75 | 143 | 151 | 91.82 | 174.00 | 169.36 | 10.73 | 22.11 | 16.060 |
| Medium 2 | 61 | 148 | 148 | 70.18 | 168.09 | 160.90 | 6.43 | 16.38 | 10.870 |
| Medium 3 | 124 | 191 | 174 | 137.36 | 211.63 | 187.45 | 9.59 | 16.44 | 10.650 |
| Medium 4 | 69 | 152 | 137 | 74.27 | 168.00 | 150.09 | 5.16 | 12.77 | 8.083 |
| Medium 5 | 96 | 158 | 121 | 117.45 | 181.45 | 145.18 | 12.32 | 11.81 | 16.190 |
| Large | 634 | 772 | 734 | 660.27 | 838.90 | 808.00 | 20.86 | 34.47 | 23.740 |

For example, if we consider an individual comparison with the results obtained in Medium 4 dataset, we can see that AAC algorithm rank is 9 over all 26 approaches with the percentage deviation of 0.96, where all the approaches that outperform AAC algorithm in medium 4 dataset are better for their structure such as a population based approaches or hybridization approaches or intelligent neighborhood selection, in addition that they employs many neighborhood structure, while AAC employ two neighborhoods structure. Where, in A3 (Abdullah *et al*., 2007) is a hybridize mechanism for great deluge, A15 (Turabieh and Abdullah, 2009), A18 (Al-Betar *et al*., 2010), A19 (Turabieh *et al*., 2010), A20 (Jaradat and Ayob, 2010) and A26 (Abuhamdah *et al*., 2013) are a population based algorithm and some of them hybrid approaches, in A24 (Abuhamdah and Ayob, 2010) it is also hybridization with adaptive mechanism and in A25 (Abuhamdah and Ayob, 2010) employ hybridization and systemic neighborhood selection. However, if we compare the results in Medium 2 dataset, then we can see that AAC ranked as 4 over all with the percentage deviation of 0.56. Where, AAC outperformed by A19 (Turabieh *et al*., 2010), A25 (Abuhamdah and Ayob, 2010) and A26 (Abuhamdah *et al*., 2013); and the reason behind that A19 and A26 are a population based approaches with hybridization and employs many neighborhood structure, where A25 is a hybridization with intelligent neighborhood selection. This poses a future work, to employ more neighborhood structures with intelligent selection, which may help in producing better solution and outperform other approaches.

Figure 5 shows the box and whisker plot that summarize the results of 11 runs on Socha benchmark datasets (note that in the clustering problems results, the minimal distances difference are so high, therefore, AAC behaviour can be more understandable in timetabling problem).

In Fig. 5 the results for the small datasets are obtained between 283 to 1,299 seconds (for more details see Table 15). Meanwhile, the medium and large datasets ranged from 19,152 to 35,744 seconds.

In all datasets, we can see that the median is slightly closer to the best than to the worst of these runs. This indicates that the algorithm is stable and consistent most of the time and may produce very good quality solutions. The result also shows that AAC is capable of producing feasible solution for all datasets with high quality solutions that are comparable with the best-known results obtained in the literature.

For example of AAC behaviour on the medium 1 dataset (see Fig. 6). Figure 6 shows the correlations between the minimize number of iterations to 200 iterations and the solution quality (or penalty) to be more understandable for observing AAC algorithm performance.

Table 14. Comparison between AAC algorithm and other approaches in the literature using the eleven datasets for course timetabling problem

| Data set | Rank | Δ(%) | AAC | A1 | A2 | A3 | A4 | A5 | A6 | A7 | A8 | A9 | A10 | A11 | A12 | A13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *Small1* | Same | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 10 | 2 | 0 | 6 | 3 | 1 | 1 | 8 |
| *Small2* | Same | 0 | 0 | 0 | 0 | 0 | 0 | 5 | 9 | 4 | 0 | 7 | 4 | 3 | 2 | 11 |
| *Small3* | Same | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 7 | 2 | 0 | 3 | 6 | 1 | 0 | 8 |
| *Small4* | Same | 0 | 0 | 0 | 0 | 0 | 0 | 3 | 17 | 0 | 0 | 3 | 6 | 1 | 1 | 7 |
| *Small5* | Same | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 7 | 4 | 0 | 4 | 0 | 0 | 0 | 5 |
| *Medium1* | 7 | 0.83 | 75 | 317 | 175 | 80 | 221 | 176 | 243 | 254 | 242 | 372 | 140 | 195 | 146 | 199 |
| *Medium2* | 4 | 0.56 | 61 | 313 | 197 | 105 | 147 | 154 | 225 | 258 | 161 | 419 | 130 | 184 | 173 | 202.5 |
| *Medium3* | 8 | 1.07 | 124 | 357 | 216 | 139 | 246 | 191 | 249 | 251 | 265 | 359 | 189 | 248 | 267 | - |
| *Medium4* | 9 | 1.16 | 69 | 247 | 149 | 88 | 165 | 148 | 285 | 321 | 181 | 348 | 112 | 164.5 | 169 | 177.5 |
| *Medium5* | 9 | 0.96 | 96 | 292 | 190 | 88 | 130 | 166 | 132 | 276 | 151 | 171 | 141 | 219.5 | 303 | - |
| *large* | 8 | 0.56 | 634 | 926 | 912 | 730 | 529 | 798 | 1138 | 1027 | - | 1068 | 876 | 851.5 | 1166 | - |
| Data Set | AAC | A14 | A15 | A16 | A17 | A17 | A18 | A19 | A20 | A21 | A22 | A23 | A24 | A25 | A26 | |
| *Small1* | 0 | 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| *Small2* | 0 | 3 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| *Small3* | 0 | 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| *Small4* | 0 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| *Small5* | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| *Medium1* | 75 | 316 | 55 | 126 | 71 | 88 | 168 | 45 | 84 | 117 | 105 | 82 | 64 | 51 | 41 | |
| *Medium2* | 61 | 243 | 70 | 123 | 82 | 88 | 160 | 40 | 82 | 108 | 108 | 78 | 65 | 54 | 39 | |
| *Medium3* | 124 | 255 | 102 | 185 | 137 | 112 | 176 | 61 | 123 | 135 | 156 | 136 | 91 | 95 | 60 | |
| *Medium4* | 69 | 235 | 32 | 116 | 55 | 84 | 144 | 35 | 62 | 75 | 84 | 73 | 66 | 48 | 39 | |
| *Medium5* | 96 | 215 | 61 | 129 | 106 | 103 | 71 | 49 | 75 | 160 | 141 | 103 | 89 | 75 | 55 | |
| *large* | 634 | - | 653 | 821 | 777 | 915 | 417 | 407 | 690 | 589 | 719 | 680 | 576 | 609 | 463 | |

Note: A1: (Abdullah *et al*., 2005. A2: (Abdullah and Turabieh, 2009). A3: (McMullan, 2007). A4: (Abdullah *et al*., 2007). A5: (Ejaz and Javed, 2007). A6 (Asmuni *et al*., 2005). A7: (Abdullah and Turabieh, 2008). A8: (Abdullah *et al*., 2007). A9: (Burke *et al*., 2007). A10: (Landa-Silva and obit, 2008). A11: (Socha *et al*., 2002). A12: (Burke *et al*., 2003). A13: (Socha *et al*., 2003). A14: (Al-Betar *et al*., 2008). A15: (Turabieh and Abdullah, 2009). A16: (Landa-Silva and obit, 2009). A17: (Obit *et al*., 2009). A18: (Al-Betar *et al*., 2010). A19: (Turabieh *et al*., 2010). A20: (Jaradat and Ayob, 2010). A21: (Shaker and Abdullah, 2010). A22: (Abuhamdah and Ayob, 2009). A23: (Abuhamdah and Ayob, 2010). A24: (Abuhamdah and Ayob, 2010). A25: (Abuhamdah and Ayob, 2010). A26: (Abuhamdah *et al*., 2013).
*The value for AAC and A1-A26 are the minimum penalty cost obtained by each approach.*

Table 15. Statistical analysis of AAC algorithm applied to Socha benchmark datasets

| Dataset | *fmin* | Iterations | Time/s |
|---|---|---|---|
| Small 1 | 0 | 31,311 | 346.58 |
| Small 2 | 0 | 158,086 | 1009.79 |
| Small 3 | 0 | 60,641 | 287.60 |
| Small 4 | 0 | 34,008 | 181.38 |
| Small 5 | 0 | 10,812 | 298.49 |
| Medium 1 | 75 | 151,346 | 29139.00 |
| Medium 2 | 61 | 152,414 | 30495.00 |
| Medium 3 | 124 | 166,142 | 32374.00 |
| Medium 4 | 69 | 117,529 | 25835.00 |
| Medium 5 | 96 | 134,942 | 28581.00 |
| large | 634 | 173,852 | 34148.00 |

In Fig. 6 the curve slope shows that when the number of iterations increase, then the penalty cost improved. In the beginning of the search, we can see that the penalty cost can be quickly reduced when the worse solution accepted, which show a flexible acceptance criteria (i.e., AC). In addition that, accepting the worst solution in AAC is capable of escaping from local optima and acceptance a little worse solution from the beginning to the end of the curve became more smaller, which may help the search to produce a better quality solutions. For more understanding about AAC algorithm in producing good quality solutions, the following table illustrates the statistical analysis of applying AAC algorithm on the Socha benchmark datasets. The statistical readings are based on the following performance indicators: The best score (*fmin*), the total number of iteration moves (Iterations) for the best solution and the total CPU time on the computer needed to find the best solution *fmin* (Time/s).

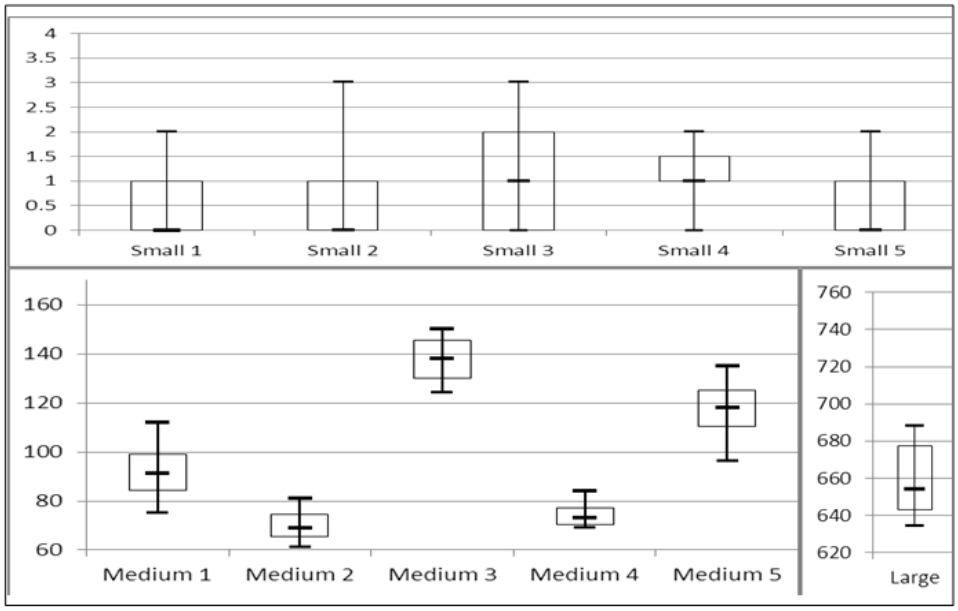


Fig. 5. Box and whisker plot of AAC for all datasets of the course timetabling problem

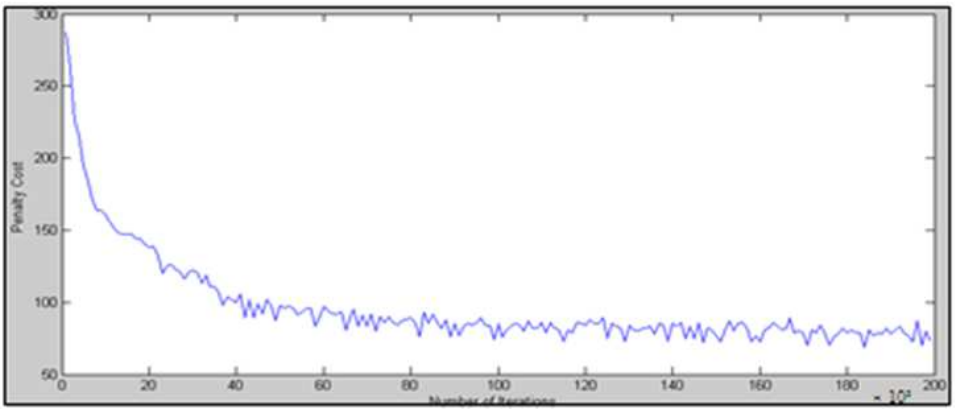


Fig. 6. AAC algorithm behavior of course timetabling problem on the medium 4 dataset

If there are multiple hits on the best solution in each independent run (11 runs for small, 11 runs for medium and 11 runs for the large instance), then the values listed in the table are the average over these multiple best hits (see Table 15). However, the average scores (favg) and the standard deviation σ for AAC algorithm is shown in Table 12.

## Conclusion and Discussion

This work has proposed a non-parametric Acceptance Criterion (AC) that not relies on user-defined that is motivated by weakness of the local search algorithms in tuning the parameters. However, the other limitation of many local based algorithms is in exploring the search space (diversification strategy), which motivates to increase diversification strategy in AC by employing a similar idea of an adaptive mechanism previously proposed in ARDA algorithm to overcome the other limitation of the local based algorithm and termed as AAC algorithm.

In order to evaluate the AC and AAC algorithms performance, AC and AAC are tested on two problem domain for optimization, the first domain problem is six medical clustering benchmark dataset and the second is eleven benchmark datasets for university course timetabling problem. A comparison made between the performance of AC, AAC, SA, GD and other approaches in the literature. Results indicate that AC is a good acceptance criterion as it outperformed SA, GD in some datasets for both domains. Results also shows that, AAC outperform AC, SA and GD algorithms in both domain problems in most datasets, outperformed other methods in the literature for the clustering problem in most datasets; and obtain comparable results with other

methods in the literature for the course timetabling problem and the results are generally statistically significant comparing to those methods. Thus we can conclude that AAC has more capability in intensifying and diversifying the search than AC. The limitation of AC approach is that we need to increase the diversification, where the limitation in AAC is to control the diversification intelligently with better mechanism of diversification or hybridize it with a good mechanism. The limitation in both AC and AAC algorithms is in the neighborhood selection, which needs to be selected intelligently and more investigation on increasing the number of neighborhood structure for its role in the results quality.

## Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved.

## References

Abdullah, S. and H. Turabieh, 2008. Generating university course timetable using genetic algorithms and local search. Proceedings of the 3rd International Conference on Convergence and Hybrid Information Technology, Nov. 11-13, IEEE Xplore Press, Busan, pp: 254-260. DOI: 10.1109/ICCIT.2008.379

Abdullah, S., H. Turabieh, 2009. Electromagnetic like mechanism and great deluge for course timetabling problems. Proceedings of the 1st Seminar on Data Mining and Optimization (DMO' 09).

Abdullah, S. Burke, E.K. and B. Mccollum, 2005. An investigation of variable neighbourhood search for university course timetabling. Proceedings of the 2nd Multidisciplinary Conference on Scheduling: Theory and Applications, Jul. 18-21, ASAP Research Group, NY, USA, pp: 413-427.

Abdullah, S., E.K. Burke and B. McCollum, 2007. A hybrid evolutionary approach to the university course timetabling problem. Proceedings of the IEEE Congress on Evolutionary Computation, Sep. 25-28, IEEE Xplore Press, Singapore, pp: 1764-1768. DOI: 10.1109/CEC.2007.4424686

Abdullah, S., E.K. Burke and B. McCollum, 2007. Using a Randomised Iterative Improvement Algorithm with Composite Neighbourhood Structures for the University Course Timetabling Problem. In: Metaheuristics. Doerner, K.F., M. Gendreau, P. Greistorfer, W. Gutjahr and R.F. Hartl *et al*. (Eds.), Springer, USA. ISBN-10: 978-0-387-71919-1, pp: 153-169.

Abuhamdah, A. and M. Ayob, 2010. Adaptive randomized descent algorithm for solving course timetabling problems. Int. J. Phys. Sci., 5: 2516-2522.

Abuhamdah, A. and M. Ayob, 2010. Adaptive randomized descent algorithm using round robin for solving course timetabling problems. Proceedings of the 10th International Conference on Intelligent Systems Design and Applications, Nov. 29-Dec. 1, IEEE Xplore Press, Cairo, pp: 1201-1206. DOI: 10.1109/ISDA.2010.5687021

Abuhamdah, A. and M. Ayob, 2010. Multi-neighbourhood particle collision algorithm for solving course timetabling problems. Proceedings of the 2nd Conference on Data Mining and Optimization, Oct. 27-28, IEEE Xplore Press, Kajand, pp: 21-27. DOI: 10.1109/DMO.2009.5341917

Abuhamdah, A. and M. Ayob, 2011. MPCA-ARDA for solving course timetabling problems. Proceedings of the 3rd Conference on Data Mining and Optimization, Jun. 28-29, IEEE Xplore Press, Putrajaya, pp: 171-177. DOI: 10.1109/DMO.2011.5976523

Abuhamdah, A., 2012. Modified great deluge for medical clustering problems. Int. J. Emerg. Sci., 2: 345-360.

Abuhamdah, A., B.M. El-Zaghmouri, A. Quteishat and R. Kittaneh, 2012. Hybridization between iterative simulated annealing and modified great deluge for medical clustering problems. World Comput. Sci. Inform. Technol. J., 2: 131-136.

Abuhamdah, A., H. Turabieh, M. Abu-Zanona and Y. Khaleel, 2014. Adaptive Great Deluge (AGD) for medical clustering problem. Int. J. Emerg. Sci., 4: 1-13.

Abuhamdah, A., M. Ayob, 2009. Experimental result of particle collision algorithm for solving course timetabling problems. Int. J. Comput. Sci. Netw. Security, 9: 134-142.

Abuhamdah, A., M. Ayob, G. Kendall and N. Sabar, 2013. Population based Local Search for university course timetabling problems. Applied Intelligence J., 40: 44-53. DOI: 10.1007/s10489-013-0444-6

Al-Betar, M.A., A.T. Khader and I.Y. LIAO, 2010. A Harmony Search with Multi-Pitch adjusting rate for the University Course Timetabling. In: Recent Advances in Harmony Search Algorithm, Geem, Z.W. (Ed.), Springer Berlin Heidelberg, ISBN-10: 978-3-642-04316-1, pp: 147-161.

Al-Betar, M.A., A.T. Khader and T.A. Gani, 2008. A harmony search algorithm for university course timetabling. Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling, Aug. 19-22, Montreal, Canada, pp: 18-22.

Asmuni, H., E.K. Burke and J.M. Garibaldi, 2005. Fuzzy multiple heuristic ordering for course timetabling. Proceedings of the 5th United Kingdom Workshop on Computational Intelligence, Sep. 5-7, ASAP Research Group, London, UK, pp: 302-309.

Ayvaz, D., H. Topcuoglu and F. Gurgen, 2012. Performance evaluation of evolutionary heuristics in dynamic environments. Applied Intelligence, 37: 130-144. DOI: 10.1007/s10489-011-0317-9

Berry, M.J. and G.S. Linoff, 1997. Data Mining Techniques: For Marketing, Sales and Customer Support. 1st Edn., John Wiley and Sons, NY, USA, ISBN-10: 978-0-471-17980-1, pp: 464.

Brucker, P., 1978. On the Complexity of Clustering Problems. In: Optimization and Operations Research, Henn, R. B. Korte and W. Oettli (Eds.), Springer Berlin Heidelberg, ISBN-10: 978-3-540-08842-4, pp: 45-54.

Burke, E.K., B. McCollum, A. Meisels, S. Petrovic and R. Qu, 2007. A graph-based hyper-heuristic for educational timetabling problems. Eur. J. Operational Res., 176: 177-192. DOI: 10.1016/j.ejor.2005.08.012

Burke, E.K., G. Kendall and E. Soubeiga, 2003. A tabu-search hyperheuristic for timetabling and rostering. J. Heuristics, 9: 451-470. DOI: 10.1023/B:HEUR.0000012446.94732.b6

Carter, M.W. and G. Laporte, 1998. Recent developments in practical course timetabling. Proceedings of the 2nd International Conference on the Practice and Theory of Automated Timetabling II, Aug. 20-22, 1997, Toronto, Canada, Springer Berlin Heidelberg, pp: 3-19. DOI: 10.1007/BFb0055878

Chiarandini, M., M. Birattari, K. Socha and O. Rossi-Doria, 2006. An effective hybrid algorithm for university course timetabling. J. Scheduling, 9: 403-432. DOI: 10.1007/s10951-006-8495-8

Dasgupta, S. and Y. Freund, 2009. Random projection trees for vector quantization. IEEE Trans. Inform. Theory, 55: 3229-3242. DOI: 10.1109/TIT.2009.2021326

Davidson, I. and A. Satyanarayana, 2003. Speeding up k-means clustering by bootstrap averaging. Proceedings of the 3rd IEEE International Conference on Data Mining Workshop on Clustering Large Data Sets, Nov. 19-22, Florida, pp: 16-25.

Ejaz, N. and M.Y. Javed, 2007. A hybrid approach for course scheduling inspired by die-hard co-operative ant behavior. Proceedings of the IEEE International Conference on Automation and Logistics, Aug. 18-21, IEEE Xplore Press, Jinan, pp: 3095-3100. DOI: 10.1109/ICAL.2007.4339114

Holland, J., 1975. Adaptation in Natural and Artificial Systems. 1st Edn., University of Michigan Press, Ann Arbor, ISBN-10: 0472084607, pp: 183.

Hong, S., 2006. Experiments with K-means, fuzzy C-means and approaches to choose K and C. Ph.D. Thesis, University of Central Florida, Florida.

Jain, A., M. Murty and P. Flynn, 1999. Data clustering: A review. J. ACM Comput. Surveys, 31: 264-323. DOI: 10.1145/331499.331504

Jaradat, G.M. and M. Ayob, 2010. An Elitist-Ant System for Solving the Post-Enrolment Course Timetabling Problem. In: Database Theory and Application, Bio-Science and Bio-Technology, Zhang, Y., A. Cuzzocrea, J. Ma, K. Chung and T. Arslan *et al*. (Eds.), Springer Berlin Heidelberg, ISBN-10: 978-3-642-17621-0, pp: 167-176.

Kittaneh, R., S. Abdullah and A. Abuhamdah, 2012. Iterative simulated annealing for medical clustering problems. Trends Applied Sci. Res. J., 2: 103-117. DOI: 10.3923/tasr.2012.103

Landa-Silva, D. and J.H. Obit, 2008. Great deluge with non-linear decay rate for solving course timetabling problems. Proceedings of the 4th International IEEE Conference Intelligent Systems, Sep. 6-8, IEEE plore Press, Varna, pp: 8-18, DOI: 10.1109/IS.2008.4670447

Landa-Silva, D. and J.H. Obit, 2009. Evolutionary Non-Linear Great Deluge for University Course Timetabling. In: Hybrid Artificial Intelligence Systems, Corchado, E., X. Wu, E. Oja and B. Baruque (Eds.), Springer Berlin Heidelberg, Spain, ISBN-10: 978-3-642-02318-7, pp: 269-276.

Lewis, R., 2008. A survey of metaheuristic-based techniques for university timetabling problems. OR Spectrum, 30: 167-190. DOI: 10.1007/s00291-007-0097-0

Maulik, U. and S. Bandyopadhyay, 2000. Genetic algorithm-based clustering technique. Patt. Recognition, 33: 1455-1465. DOI: 10.1016/S0031-3203(99)00137-5

Mcmullan, P., 2007. An extended implementation of the great deluge algorithm for course timetabling. Proceedings of the 7th International Conference Computational Science, May 27-30, Springer, Beijing China, pp: 538-545. DOI: 10.1007/978-3-540-72584-8

Mcmullan, P., 2007. An Extended Implementation of the Great Deluge Algorithm for Course Timetabling. In: Computational Science (ICCS '07), Shi, Y., G.D. van Albada (Eds.), Springer Science and Business Media, Berlin, New York, ISBN-10: 3540725830, pp: 538-545.

Obit, J.H., D. Landa-Silva, D. Ouelhadj and M. Sevaux, 2009. Non-linear great deluge with learning mechanism for solving the course timetabling problem. Proceedings of the 8th Metaheuristics International Conference, Jul. 13-16, Hamburg, Germany, pp: 1-10.

Petrovic, S. and E.K. Burke, 2004. Educational Timetabling. In: Handbook of Scheduling: Algorithms, Models and Performance Analysis, Leung, J.Y.T. (Ed.), CRC Press, USA, ISBN-10: 0203489802, pp: 41-45.

Pirlot, M., 1996. General local search methods. Eur. J. Operational Res., 92: 493-511. DOI: 10.1016/0377-2217(96)00007-0

Saha, I., D. lewczyński, U. Maulik and S. Bandyopadhyay, 2010. Consensus multiobjective differential crisp clustering for categorical data analysis. Rough Sets Current Trends Comput. LNCS, 6086: 30-39. DOI: 10.1007/978-3-642-13529-3_5

Schaerf, A., 1999. A survey of automated timetabling. Artificial Intelligence Rev., 13: 87-127. DOI: 10.1023/A:1006576209967

Schaerf, A., 1999. Local search techniques for large high school timetabling problems. IEEE Trans. Syst. Man Cybernetics Part A: Syst. Hum., 29: 368-377. DOI: 10.1109/3468.769755

Shaker, K. and S. Abdullah, 2010. Controlling multi algorithms using round robin for university course timetabling problem. Proceedings of the International Conferences Database Theory and Application, Bio-Science and Bio-Technology, Dec. 13-15, Springer Berlin Heidelberg, Jeju Island, Korea, pp: 47-55. DOI: 10.1007/978-3-642-17622-7_6

Socha, K., J. Knowles and M. Sampels, 2002. A Max-Min Ant System for the University Course Timetabling Problem. In: Ant Algorithms, Dorigo, M., G.D. Caro and M. Sampels (Eds.), Springer Berlin Heidelberg, Belgium, ISBN-10: 978-3-540-44146-5, pp: 1-13.

Socha, K., M. Sampels and M. Manfrin, 2003. Ant algorithms for the University Course Timetabling Problem with Regard to the State-of-the-Art. In: Applications of Evolutionary Computing, Cagnoni, S., C.G. Johnson, J.J.R. Cardalda, E. Marchiori and D.W. Corne *et al*. (Eds.), Springer Berlin Heidelberg, UK, ISBN-10: 978-3-540-00976-4, pp: 334-345.

Tripathy, A., 1980. A lagrangean relaxation approach to course timetabling. J. Operational Res. Society, 31: 599-603. DOI: 10.2307/2580848

Turabieh, H. and S. Abdullah, 2009. Incorporating tabu search into memetic approach for enrolment-based course timetabling problems. Proceedings of the 2nd Conference on Data Mining and Optimization, Oct. 27-28, IEEE Xplore Press, Kajand, pp: 115-119. DOI: 10.1109/DMO.2009.5341901

Turabieh, H., S. Abdullah, B. McCollum and P. McMullan, 2010. Fish swarm intelligent algorithm for the course timetabling problem. Proceedings of the 5th International Conference Rough Set and Knowledge Technology, Oct. 15-17, Springer Berlin Heidelberg, Beijing, China, pp: 588-595. DOI: 10.1007/978-3-642-16248-0_80

Wang, X.Y., 2007. Fuzzy clustering in the analysis of Fourier transform infrared spectra for cancer diagnosis. Ph.D. Thesis, School of Computer Science and Information Technology, University of Nottingham.

Zhang, T., 2001. Convergence of large margin separable linear classification. Adv. Neural Inform. Processing Syst., 13: 357-363.