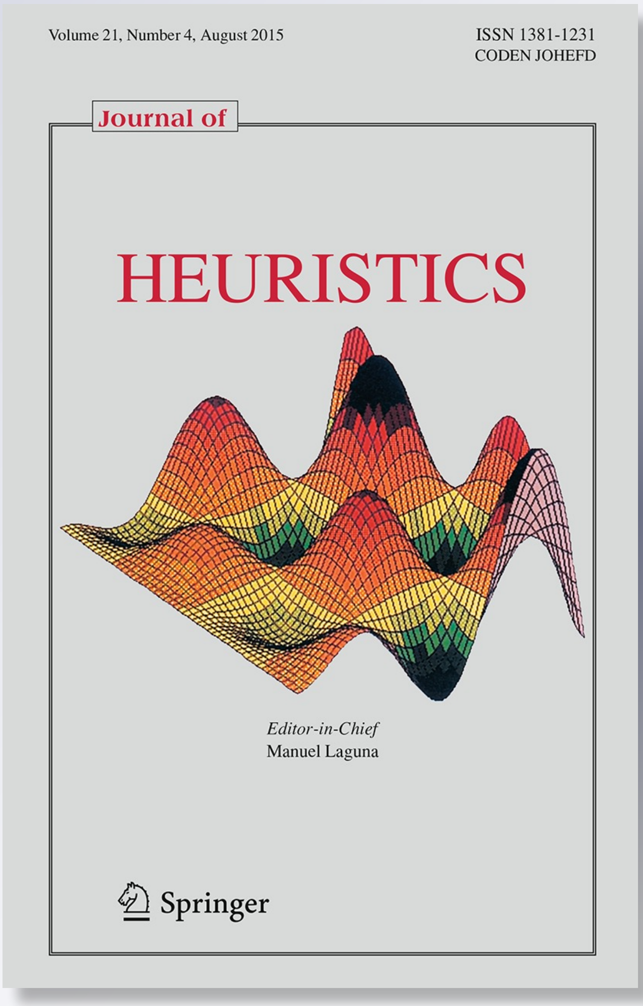
A tabu search algorithm for cohesive clustering problems

Buyang Cao, Fred Glover & Cesar Rego

Journal of Heuristics

ISSN 1381-1231
Volume 21
Number 4

J Heuristics (2015) 21:457-477
DOI 10.1007/s10732-015-9285-2



J Heuristics (2015) 21:457–477
DOI 10.1007/s10732-015-9285-2

A tabu search algorithm for cohesive clustering problems

Buyang Cao · Fred Glover · Cesar Rego

Received: 11 October 2014 / Revised: 23 January 2015 / Accepted: 27 January 2015 /
Published online: 12 February 2015

Abstract Clustering problems can be found in a wide range of applications including data mining/analytics, logistics, healthcare, biotechnology, economic analysis and many other areas. Solving a clustering problem from the real world often poses significant challenges in spite of the fact that extensive research has been devoted to this topic. In this paper we present a tabu Search algorithm for a new problem class called cohesive clustering which arises in a variety of business applications. The class introduces an objective function to produce clusters as "pure" as possible, to maximize the similarity of the elements in each given cluster. Tabu search intensification and diversification strategies are employed in order to produce enhanced outcomes. The computational results demonstrate the effectiveness of the proposed algorithm.

Keywords Clusters · Heuristic · Meta-heuristic · Tabu search

1 Introduction

Clustering is commonly defined to consist of grouping a set of objects in a manner that causes objects in the same group or cluster to be more similar (or closer) to

B. Cao (✉)
China Intelligent Urbanization Co-Creation Center for High Density Region,
School of Software Engineering, Tongji University, 4800 Cao'An Road, Shanghai 201804, China
e-mail: caobuyang@tongji.edu.cn

F. Glover
OptTek Systems, Inc., 2241 17th Street, Boulder, CO 80302, USA
e-mail: glover@opttek.com

C. Rego
School of Business Administration, University of Mississippi, University, MS 38677, USA
e-mail: crego@bus.olemiss.edu

each other than they are to objects in other groups or clusters. Clustering problems arise in many applications. Within the realm of business applications, which provides the setting for the present work, a common type of clustering problem (April et al. 2014) arises in the situation where organizations seek to aggregate different employees into "employee pools" whose members share significant similarities and attributes. A related application discussed in the same reference arises in settings where it is desired to group jobs into common categories so that each category can be treated as a unified job classification. Linoff and Berry (2011) illustrate an application where the clustering challenge is to build customer segments for a publishing company. In this case the company is interested in finding segments with particularly good customers. Provost and Fawcett (2013) note that clustering is very often used by business units to target their marketing segments.

To handle business applications more effectively, a variety of clustering methods have been developed that are tailored specifically for problems found in business contexts. Liu (1999) proposes a clustering-based algorithm to cluster and arrange items for stock location and pick-up operations for a warehouse application. Two similarity measures are proposed for items and customers according to order quality and types of items. Simulation indicates the proposed model and solution methodology are capable of creating useful results for its area of application.

Grabmeier and Rudolph (2002) conduct a survey on clustering techniques in data mining business, focusing on clustering models where the goal is to create clusters based upon (1) similarities or similarity indices and (2) distance functions. Criteria to evaluate the clustering results are provided to assist scholars to judge the outcomes obtained by various clustering algorithms. The paper provides useful suggestions for modeling clustering problems properly and picking appropriate solution methodologies and optimization criteria.

Strehl and Ghosh (2002) develop an algorithm to address real-life data-mining problems found in retail-industry applications, in which data reside in a high dimensional space. Their approach introduces a similarity relationship defined on each pair of data samples that transforms the problem into one over the similarity domain without the original high-dimensional space. The goal is to cluster data samples into k groups so that data samples for different clusters have similar characteristics. The problem is formulated as a vertex-weighted graph partitioning problem, where each vertex (data sample) is assigned a weight representing its importance and each pair of vertices is connected by an undirected edge whose weight is determined by their similarity measurement. The objective of this partitioning problem is to produce a minimum weight solution that accounts for the vertex weight balancing constraint. An algorithm called OPOSSUM (optimal partitioning of space similarities using metis) is developed to tackle the underlying problem.

Kochenberger et al. (2005) introduce a variant of clustering based on a clique partitioning model to analyze gene data more effectively. The authors propose a tabu search based approach and perform a computational study that documents its effectiveness for the problem class considered.

The well-known K-means algorithm is applied widely in solving clustering problems in business settings with adaptations for the case at hand. For example, in order to accommodate a distributed data environment Datta et al. (2009) propose a distributed

K-means clustering approach over a peer-to-peer network. The approach is scalable, so that it can be applied to solve large-scale clustering problems.

Trappey et al. (2010) build on the methodology of Liu (1999) to identify how to cluster customers into segments so that logistics companies can provide better and specialized services to different customer segments upon their preferences and demands.

Bae et al. (2010) introduce a new measure of clustering similarity to overcome shortcomings of existing similarity criteria. They point out two major issues to be addressed: inabilities to detect structural dissimilarity and the need to compare clusters of non-overlapping points. The measure proposed by the authors represents clusters as density profiles, and is able to consider feature distribution and point relationship information. The measure can be utilized to compare clusters, evaluate the clustering results, and to create alternative clusters with good qualities.

Clustering problems found in logistics applications often arise where logistics companies apply clustering models to build their service areas, taking the travel distances and/or times into account. In order to minimize travel distance or travel time the clusters are built to be as compact as possible. Cao and Glover (2010) propose an algorithm based on Thiessen-polygon capable of generating balanced and connected cluster. Computational tests on real applications demonstrate the efficiency of the proposed algorithm. Nowadays big data applications clustering problems are encountered frequently, e.g., finding communities in social networks, classifying customers, etc. Wu et al. (2013) proposed a cluster-based methodology in order to solve “cold start” problems involving collaborative recommendation more effectively. They first apply an algorithm similar to K-means to cluster customers according to their properties and then identify products/services appropriate to recommend to customers in each cluster.

In addition to the well-known K-means algorithm, researchers have developed a variety of clustering algorithms attempting to solve various clustering problems more efficiently. A useful review of this area is provided by Brusco et al. (2012), who survey a variety of non-traditional clustering algorithms ranging from model-based to metaheuristics-based methods, focusing on an objective function derived from a proximity measurement.

An important clustering application which motivates our present work arises in one of world largest e-commerce companies (a confidentially agreement prevents us from giving this company’s name). In order to detect the exceptions occurring on its cloud platforms efficiently, the company needs to identify patterns associated with the jobs performed. However, there are many jobs and it is not effective to identify the pattern for an individual job. Therefore, it is worthwhile to group jobs into several categories and characterize the patterns for job categories in relation to their attributes (for instance, CPU usages, I/O, memory requirement, etc.).

To address this and related applications we present a clustering model that incorporates an objective designed to create clusters in which elements (jobs) should be as “pure” as possible. The model is conceived from the standpoint of generating solutions that provide a baseline to classify new elements, as where it is desired to assign new jobs to the proper job segments. From this perspective, the goal is not to generate clusters rapidly but rather to create a judicious collection of clusters with the aim of using them multiple times over an extended horizon as a means for future classification.

In the following exposition, Sect. 2 describes the clustering problem and its objective function in more detail, and the algorithms to solve the clustering problem discussed here. Computational results documenting the effectiveness of our new procedures are presented in Sect. 3. Finally, we summarize our findings and present some conclusions in Sect. 4.

2 Model and algorithms

2.1 Problem description and model

Let $x = (x_1, \ldots, x_k)$ represent a vector of k-attributes (factors), each describing a specific element. We associate such a vector $x(t)$ with *tth* element so that the problem data consists of a collection of such vectors represented by

$$x(t), t = 1 \; to \; N_P \tag{2.1}$$

We use N_P to denote the total number of such element under consideration. For a given set of element $T = (t_1 \ldots, t_n)$ (where $n = N_p$) in which each element is described by (2.1) and n is the total number of elements in the set. Without losing generality in the following discussions we assume each element t possesses a unique type or property presented by $x(t)$. Our goal is to aggregate subsets of elements into clusters, i.e., assigning each t_i to a specific cluster where the collection of all such clusters is denoted by

$$C\,(s)\,, \text{ for } s = 1 \; to \; N_s \tag{2.2}$$

and where N_S identifies the total number of clusters under the consideration and it is predefined. As previously indicated the objective of the clustering problem we consider will be formulated to account both for the distributions (we use distribution in this paper to describe how elements are spread geographically on a “plane”) and similarities of elements to be clustered. Before stating the objective formally, some notation and definitions are required.

Let $Score(t_i, t_j)$ denote the “score” that measures the desirability or undesirability of assigning elements t_i and t_j to the same cluster. A desirability measure is relevant for maximizing while an undesirability measure is relevant for minimizing. For example, the value $Score(t_i, t_j)$ can be a form of undesirability measure in the sense of representing a form of “distance” between elements where the distance is determined upon their attributes (2.1). The score is applied to any pair of elements in T. We may view the goal in this instance as seeking to minimize distances (broadly conceived) between elements lying in the same cluster.

2.1.1 Calibration of the scores

the calibration of the values represented by (t_i, t_j) may be achieved by using various distance-related measures, for instance:

$$Score(t_i, t_j) = ||x(t_i) - x(t_j)|| \tag{2.3}$$

This measurement could be one of the following forms:

$$\sum \left(|x_p(t_i) - x_p(t_j)| : p = 1 \ to \ k\right) \qquad \text{(L1 Norm)} \tag{2.4}$$

or

$$\sqrt{\sum \left(x_p\left(t_i\right) - x_p\left(t_j\right)\right)^2 : p = 1 \ to \ k} \quad \text{(L2 Norm)} \tag{2.5}$$

Evidently, when $k = 2$ in (2.5) we have the commonly used Euclidean "distance". It may happen that some factor p in x(t) $(t = 1, \ldots, n)$ should receive a greater emphasis than other factors in determining the scores, as where a particular factor is considered to have a more substantial impact on determining the outcome of the clustering. To consider this scenario we may associate a weight w_p with each factor in vector $x(t)$, and replace each $x_p(t)$ by $w_p x_p(t)$ $(t = 1, \ldots, n)$ before computing the values of $Score(t_i, t_j)$.

It should be emphasized that other ways of defining $Score(t_i, t_j)$ are also possible in specific applications. The following section gives an approach for pre-calculating and updating $Score(t_i, t_j)$ for efficiency.

2.1.2 Evaluating clusters by referencing scores

given the evaluation measures provided by the $Score(t_i, t_j)$ matrix, we evaluate the clusters by defining the value of cluster $C(s)$ at two levels. At the first level we define:

$$V(s) = \sum \left(Score(t_i, t_j) : t_i < t_j \ \& \ \ t_i, t_j \in C(s)\right) \tag{2.6}$$

$V(s)$ may be considered as a precursor to identifying an evaluation for cluster $C(s), s = 1 \, to \, N_S$. In order to compare different ways of assigning the elements t to clusters, we account for the fact that the number of items in the sum composing $V(s)$ skews the $V(s)$ measure. We want to adjust for the number of these items in order to more appropriately evaluate $C(s)$.

A cluster $C(s)$ can be considered as a complete graph, whose number of elements we denote by $n(s)$. Each edge then may be viewed as a "link" between two elements t_i and t_j in the same cluster, and the number of these links in cluster $C(s)$ is given by:

$$NL(s) = n(s)(n(s) - 1)/2 \tag{2.7}$$

Then the evaluation for cluster $C(s)$ (called the *Full Value* in the following discussions) will be defined as follows:

$$FV(s) = V(s)/NL(s) \tag{2.8}$$

The use of *FV(s)* makes it possible to evaluate the entire collection of clusters. However since we are considering the distributions as well as the similarities of underlying elements, a situation where two elements are very "close" together doesn't mean they are similar. In a related sense, Linoff and Berry (2011) point out that in a "town clustering" project two towns that possess the similar demographics are not necessarily close to each other geographically. In order to consider both distributions and similarities of elements more effectively we consider the *variance* of a cluster defined as follows.

Based on the fact that *FV(s)* in (2.8) is actually the average score of cluster $C(s)$, the variance of cluster $C(s)$ can be determined by:

$$VAR\,(s) = \left(\sum\nolimits_{all\ t_i,t_j}\left(\left(Score\left(t_i,t_j\right) - FV(s)\right)^2\right)/NL(s)\right. \tag{2.9}$$

2.1.3 Objective function

Drawing on the observations of the preceding discussion, the objective function we propose for clustering is given by

$$ObjVal = min\left(\alpha_1 \sum\nolimits_1^{N_s} FV\,(s) + \alpha_2 \sum\nolimits_1^{N_s} VAR(s)\right) \tag{2.10}$$

where α_1 and α_2 are adjustable weights for the full value and variance, and provide flexibility to consider different balances between distributions and similarities of elements.

2.2 Basic procedures

The goal of our problem, roughly stated, is to create a clustering set to achieve the objective function stated in (2.11), following. We present some basic components of our solution methodology in this subsection.

2.2.1 Assigning elements to clusters

The assignment of elements to clusters will be recorded in two ways. First, we maintain a value

$$C\_ID\,(t)\ \text{ for } t = 1\ to\ N_p \tag{2.11}$$

where *C_ID(t)* = *s* indicates that cluster $C(s)$ is the unique cluster to which element *t* (and its associated attribute vector $x(t)$) is assigned.

Second, we propose the use of a doubly linked list

$$B\,(t)\ \text{ and } A(t) \text{ for each } t = 1 \text{ to } N_p \tag{2.12}$$

$B(t)$ indicates the element before t and $A(t)$ specifies the element after t in the same cluster. This is accompanied by an array $F(s)$ for each s, where $t = F(s)$ is the "first" element of cluster s and successive elements are located by setting $t = A(t)$ until reaching the dummy element $t = 0$. The following operation enumerates all elements in a cluster:

```
//Procedure: loop all elements
$t = F(s)$
While $t > 0$
    $t = A(t)$
Endwhile
```

Initialization occurs by setting $F(s) = 0$ for all s. Throughout all steps, $B(t) = 0$ if $F(s) = 0$, and $A(t) = 0$ if t is the last element for s, as insured by the following operations.

When an element t_0 is added to a cluster $C(s_A)$ ("A" stands for "Add"), then t_0 is made the new "first" element of $C(s_A)$ through the following procedure:

```
//Procedure: Insert an element($t_0$, $s_A$)
$t_{After} = F(s_A)$
$B(t_{After}) = t_0$
$A(t_0) = t_{After}$
$F(s_A) = t_0$
$C\_ID(t_0) = s_A$
```

The above procedure executes

$$C(s_A) = C(s_A) \cup \{t_0\} \tag{2.13}$$

2.2.2 Dropping elements from clusters

Once all elements are initially assigned to clusters by the steps described in the above section, in order to achieve more satisfactory solutions we perform operations that move selected elements t0 from one cluster to another under the control of a tabu search process. To handle the "move operations" we identify $s_0 = C\_ID(t_0)$ and drop t_0 from $C(s_0)$ by the following procedure:

```
//Procedure: Drop an element($t_0$, $s_0$)
$t_{Before} = B(t_0)$
$t_{After} = A(t_0)$
$A(t_{Before}) = t_{After}$
$B(t_{After}) = t_{Before}$
If $t_{Before} = 0$ then
    $F(s_0) = t_{After}$
Endif
```

By the convention of maintaining a "dummy cluster" $C(0)$ that contains all elements t not assigned to real cluster. In the case where an element is dropped without being transferred to another real cluster, we have $s_A = 0$.

Recall that $n(s)$ denote the number of elements in cluster $C(s)$. The preceding procedure results respectively in setting:

$$n(s_A) = n(s_A) + 1 \text{ and } n(s_0) = n(s_0) - 1 \tag{2.14}$$

The procedures discussed so far compose the basic steps for building clusters and helping evaluate the quality of a cluster built. In the following sections we present the strategy to evaluate the quality of a cluster and provide guidance to generate good clusters.

2.2.3 Updates and evaluations

The discussion of this section considers various aspects of the clustering algorithm that contribute to its efficacy.

The following discussion will be based on evaluating a move that drops an element t$_0$ from cluster $C(s_0)$, where $s_0 = C\_ID(t_0)$, and adding t_0 to another $C(s_A)$. We refer to this move as "$t_0 \rightarrow s_A$".

The move may be viewed as consisting of two parts. First is the operation of adding t_0 to s_A. We identify and store the quantity $AddVal(t_0, s_A)$ so that

$$AddVal\left(t_0, s_A\right) = \sum \left(Score\left(t_0, t\right) : t \in C\left(s_A\right)\right) \tag{2.15}$$

Note that $AddVal(t_0, s_A)$ is precisely the increase in the sum of the link scores in cluster $C(s_A)$. Hence it is the amount of increase in $V(s_A)$ (defined in (2.6)) caused by the $t_0 \rightarrow s_A$ move.

The symbol # is used to denote a new value. Hence, for the indicated move we have

$$V\#\left(s_A\right) = V\left(s_A\right) + AddVal\left(t_0, s_A\right) \tag{2.16}$$

$$NL\#\left(s_A\right) = NL\left(s_A\right) + n\left(s_A\right) \tag{2.17}$$

$$FV\#\left(s_A\right) = V\#\left(s_A\right)/NL\#\left(s_A\right) \tag{2.18}$$

Furthermore, $Var\_A(t_0, s_A)$ is defined as the variance caused by adding t_0 to s_A, which is calculated as in (2.10). This allows us to compute the "delta change," *DelA* ("delta for adding") in terms of the combination of full value and variance due to adding t_0 to s_A given by:

$$DelA = a_1{}^*\left(FV\#\left(s_A\right) - FV\left(s_A\right)\right) + a_2{}^*\left(Var\_A\left(t0, s_A\right) - Var\left(s_A\right)\right) \tag{2.19}$$

The second part consists of the procedure of dropping t_0 from s_0, where $s_0 = C\_ID(t_0)$. We similarly define the value for dropping t_0 from cluster s_0 by:

$$AddVal\left(t_0, s_0\right) = \sum \left(Score\left(t_0, t\right) : t \in C\left(s_0\right)\ and\ t \neq t_0\right) \tag{2.20}$$

We then have:

$$V\#\left(s_0\right) = V\left(s_0\right) - AddVal\left(t_0, s_0\right) \tag{2.21}$$

$$NL\#\left(s_0\right) = NL\left(s_0\right) - \left(n\left(s_0\right) - 1\right) \tag{2.22}$$

$$FV\#\left(s_0\right) = V\#\left(s_0\right)/NL\#\left(s_0\right) \tag{2.23}$$

Accordingly we define $Var\_D(t_0, s_0)$ as the variance of cluster s_0 or $C(s_0)$ after element t_0 is dropped. Thus we have the "delta value," *DelD*, for dropping t_0 from s_0:

$$DelD = a_1{}^* \left(FV\# (s_0) - FV (s_0)\right) + a_2{}^* \left(Var\_D (t_0, s_0) - Var (s_0)\right) \quad (2.24)$$

Based on these values the change in the objective function value, *DelObjVal* as a result of both adding t_0 to s_A and dropping t_0 from s_0 is given by

$$DelObjVal = DelA + DelD \quad (2.25)$$

Under a minimization objective, $DelObjVal < 0$ identifies an improving move while $DelObjVal >= 0$ identifies a non-improving move, where the new *ObjVal* resulting from the move is given by:

$$ObjVal\# = ObjVal + DelObjVal.$$

Since (2.25) is able to recognize if a move produces an improved solution, we refer to the quantity identified in (2.25) as the *move value* which is used to evaluate a move $t_0 \to s_A$ and denote this quantity for a given pair (t_0, s_A) by $MoveVal(t_0, s_A)$.

2.2.4 Implementation of move value

When all elements are assigned to clusters (according to criteria discussed later), *FV(s)* and *VAR(s)* will be available for all s, having been computed incrementally while the clusters are being built. Based upon the calculation for $MoveVal(t_0, s_A)$ discussed above, we pre-compute and store $AddVal(t_0, s_A)$ and $AddVal(t_0, s_0)$ for all t_0, s_A, and s_0. To evaluate a move $t_0 \to s_A$ i, where element t_0 is dropped from cluster $C(s_0)$ and added to cluster $C(s_A)$, the following procedure yields the corresponding $MoveVal(t_0, s_A)$.

```
//Procedure: Updating move value
For t = 1 to $N_p$
    //consider dropping t from $s_0$ (to select $t = t_0$)
    $tmpVal = AddVal(t,s_0) – Score(t, t_0)$
    $AddVal(t, s_0) = tmpVal$
    // consider adding t to $s_A$ (to select $t = t_0$)
    $tmpVal = AddVal(t, s_A) + Score(t, t_0)$
    $AddVal(t, s_A) = tmpVal$
    Update $MoveVal(t, s_0)$ and $MoveVal(t, s_A)$ by (2.25)
Endfor
// Note: the foregoing holds for $t = t_0$ since $Score(t_0, t_0)=0$.
Update $MoveVal(t_0, s)$ by (2.25), $s = 1 \ldots N_s$.
```

2.3 Build initial solution

For a given set of elements and a given number of clusters to be built, we now describe how to construct the initial solution, i.e., to assign all elements to the clusters while achieving a reasonable value of (2.11). The procedure to build the initial solution sets α_2 to be zero in order to eliminate some undesirable scenarios, as where some elements "far away" from each other might be grouped into the same cluster due to a small variance. The variance is instead introduced during the improvement phase.

2.3.1 Creating 2-element clusters

As discussed above the first portion of the objective function relies on the value of $Score(t_0,\ t_1)$ for any t_0, t_1 in the same cluster. In order to pick two elements (called *seeds*) quickly and still create initial clusters with good quality, the following strategy called *2-element selection* is employed.

For each t_0 that is not clustered we identify the 3 best (minimum) scoring matches $Score(t_0, t_a)$, $Score(t_0,\ t_b)$, and $Score(t_0,\ t_c)$ (which can be pre-calculated and arranged), where t_0, t_a, t_b, and t_c are all distinct. Then two elements t_1 and t_2 will be selected that minimize (2.26) over the three options, i.e., $(t_1, t_2) = (t_a, t_b), (t_a, t_c), (t_b, t_c)$. Finally we select t_1 to form a cluster with the first pair of elements (t_0, t_1), where $Score(t_0, t_1) = min(Score(t_0,\ t_1), Score(t_0,\ t_2))$.

$$Score\,(t_0, t_i) + Score\left(t_0, t_j\right) + Score\left(t_i, t_j\right) \tag{2.26}$$

```
// Procedure : Create 2-element clusters
    For s = 1 to $N_s$
        bestVal = BIG
        For each $t_0$ in C(0)
            select $t_1$ and $t_2$ using the 2-element selection strategy
            tmpVal = Score($t_0$,$t_1$) + Score($t_0$,$t_2$) + Score($t_1$,$t_2$)
            If (tmpVal <bextVal)
                bestElem1 = $t_0$
                bestElem2 = $t_1$
            Endif
        Endfor
        Insert an element(bestElem1, s)
        Insert an element(bestElem2, s)
        Drop an element(bestElem1, 0)
        Drop an element(bestElem2, 0)
    Endfor
```

The number of elements $n(s)$ in each cluster $C(s)$ is required to satisfy lower and upper bounds denoted by *LB(s)* and *UB(s)*. The procedure to generate the initial solution may be described as follows.

```
//Procedure: Create initial solution
Create 2-element clusters
//fill elements to attain the low bound of each cluster
For s = 1 to $N_s$
        If (n(s) < LB(s))
                Do
                        Find an element $t_0$ that minimizes MoveVal($t_0$, s)
                        Insert an element($t_0$, s)
                        Drop an element ($t_0$, 0)
                Until (n(s) >= LB(s))
        Endif
Endif
$t_0$ = F(0)
While ($t_0$ is not null)
        bestVal = BIG
        For s = 1 to Ns
                If (n(s) < UB(s))
                        If (MoveVal($t_0$, s) < bestVal)
                                bestVal = moveVal($t_0$, s)
                                bestCluster = s
                        Endif
                Endif
        Endfor
        Insert an element($t_0$, bestCluster)
        Drop an element($t_0$, 0)
        $t_0$ = A($t_0$)
Endwhile
For all cluster computing (2.8) and (2.10) with original $\alpha_1$ and $\alpha_2$
Computing (2.11)
```

Although not explicitly stated, the solver dynamically updates *AddVal(t, s)* for each element t and cluster $C(s)$, and uses *AddVal(t, s)* to obtain *MoveVal(t, s)*. After the initial solution procedure, all elements will be assigned (we assume that the total capacity of all clusters is big enough to permit this).

2.4 Tabu Search based improvement procedure

To achieve the best value of (2.10), we introduce an improvement procedure applied to the initial solution created in the previous section, which incorporates the move $t_0 \rightarrow s_A$. $MoveVal(t_0, s_A)$ defined in (2.26) is used to judge whether a move is worthwhile.

Because the underlying problem does not require results in real time, we have the luxury to devote more time with the improvement procedure to obtain better solutions. We apply the tabu Search (TS) metaheuristic to guide the improvement procedure.

(See, e.g., Glover 1986; Glover and Laguna 1997, 2013; Kochenberger et al. 2013; Wu et al. 2013.)

The following subsections discuss the tabu lists, candidate list strategy and intensification and diversification strategies used in our algorithm.

2.4.1 Tabu tenures

The tabu search used in this paper incorporates a simple design focusing on the use of recency based memory. A two-dimensional array *TabuEnd(t, s)* is employed to define the tabu status of a move $t \rightarrow s$. We define two types of tabu tenures as follows:

Small Tabu Tenure: T_{s_size} refers to $TabuEnd(t_0, s_0)$ for t_0 in $C(s_0)$. (How to set up $TabuEnd(t_0, s_0)$ will be explained later.) If $TabuEnd(t_0, s_0) \geq iter$, where *iter* denotes the current iteration, then t_0 is tabu and cannot be dropped from cluster $C(s_0)$. This small tabu tenure value is defined as follows:

$$T_{s\_size} = Min(3, N_s/3),$$
$$T_{s\_size} = Max(T_{s\_size}, 1).$$

Tabu Tenure: T_{size}, refers to $TabuEnd(t_0, s_A)$ for an element t_0 not in cluster $C(s_A)$. In this case each t_0 can be added to $N_S - 1$ different clusters other than cluster $C(s_A)$. Similarly, $TabuEnd(t_0, s_A) \geq iter$ indicates that t_0 is tabu and cannot be added to cluster $C(s_A)$.

Let N_{cand} be the number of elements in the candidate list. Then the possible number of move options will be $N_{cand} * (N_S - 1)$ excluding the case where t_0 is not permitted to drop from its cluster. Therefore we have total $(N_{cand} - T_{s\_size)} * (\mathrm{N}_S - 1)$ move options. We will make at most 50 % (and preferably at most 30 %) of these options tabu. $T_{size}(Lim)$ and $T_{size}(Pref)$ are integers used to define T_{size}, which are given by:

$$T_{size}(Lim) = 1 + (N_{cand} - T_{s\_size)}{}^*(N_S - 1)$$
$$T_{size}(Pref) = 1 + (N_{cand} - T_{s\_size)}{}^*(N_S - 1)$$

Tabu tenure is then determined by:

$$T_{size} = Max(T_{size}(Pref), 15)$$
$$T_{size} = Min\,(T_{size}, T_{size}(Lim))$$

Aspiration criterion: at times when a move $t \rightarrow s$ is tabu it may be able to yield a better solution than those previously found. In this case we allow the tabu status of this move to be overridden and thus permit the move to be performed. We define the *aspiration criterion* as follows:

Let *bestObjVal* be the current best value of the objective function (2.11) and *curObjVal* be the value obtained due to the move $t \rightarrow s$. If $curObjVal < bestObjVal$, then we declare that the aspiration criterion is met and override the move's tabu status.

2.4.2 Elite Candidate list

If all eligible elements are examined in order to find the best $t \rightarrow s$ move, the number of moves to be evaluated will be huge. Thus it is worthwhile to identify candidates

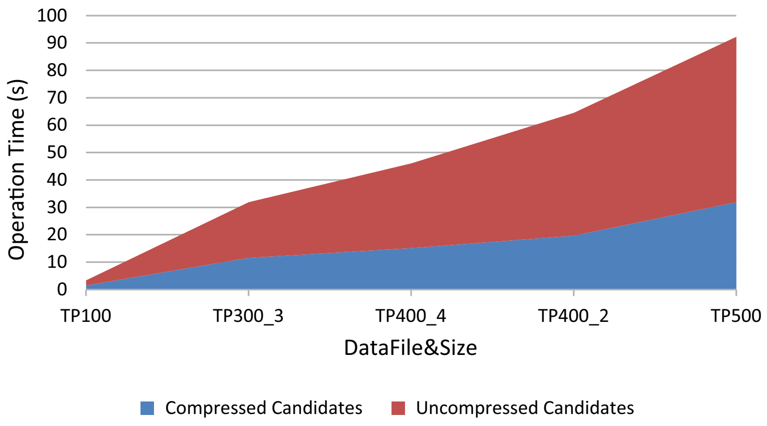


Fig. 1 The benefit of using the elite candidate list

that provide promising moves, i.e., that potentially generate good outcomes. A variety of candidate list strategies are described in Glover and Laguna (1997, 2013). Here we adopt a variation on one of these to prevent examining "poor quality" assignments for a certain number of iterations. We focus on two components, namely, (1) an element to move, and (2) the target cluster. Therefore, we use the following criterion to select candidates to be evaluated for move $t \rightarrow s$. If

$$\frac{AddVal\,(t_0, s_A)}{n\,(s_A)} < \frac{\sum \left(AddVal(t_0, s)/n(s)\right)}{N_s} \tag{2.27}$$

then move $t_0 \rightarrow s_A$ is added to an *elite candidate list* denoted by *candList*. Otherwise the corresponding move is bypassed. The underlying logic is to try to find the element for which the resulting increase in the average "distance" is smaller than the overall average increase relative to moves involving element t_0. Since *AddVal(t, s)* is computed incrementally, the value of (2.27) may be obtained quickly.

Our computational experiments disclose this strategy reduces computational time by about 50 % without causing a significant difference in the solution quality. The following picture demonstrates the benefit of applying the elite candidate list strategy, where "compressed candidates" refers to the application of the elite candidate list and TPxxx_y indicates the number of elements to be clustered in y groups (Fig. 1):

2.4.3 Intensification and diversification

During the intensification phase, the algorithm examines the moves ($t \rightarrow s$) that look promising according to the candidate list.

In order to obtain better solutions it is worthwhile to explore a broader solution space by means of a diversification strategy which we identify as follows:

Diversification: let N_P/N_S be the "baseline" number of elements in any cluster. The algorithm randomly picks elements from clusters called *source clusters* whose number of elements exceeds N_P/N_S and assigns them to clusters called *target clusters* whose number of elements is less than N_P/N_S. A source cluster $C(s)$ with $n(s) > N_P/N_S$ is randomly selected and an element t_0 is randomly selected from $C(s)$. A target cluster $C(s_A)$ with $n(s_A) < N_P/N_S$ is then randomly chosen as a basis for executing a move $t_0 \to s_A$. We assume the bounds for the number of elements in the source and target clusters are compatible with making this move. An alternative version of this approach which we plan to test in the future makes a given number of such moves, and thereafter, each time an element t_0 is chosen, selects the target cluster by sampling a specified number of candidates satisfying $n(s_A) < N_P/N_S$ and choosing the one that gives the best evaluation for making the move $t_0 \to s_A$.

2.4.4 Overall algorithm

In the following description we define a move to be feasible if this move doesn't violate the lower bound of the source cluster and the upper bound of the target cluster. The tabu search based improvement procedure can then be described as follows:

```
//Procedure: Tabu Search procedure(bestObjVal)
//initialization phase
TabuEnd(t, s) = 0 for all elements t and clusters C(s)
Compute AddVal(t, s) for all elements t and clusters C(s)
curObjVal = bestObjVal
$T_{s\_size}$ and $T_{size}$ upon the methodology described in the above section.
candList = 20% of total moves ($t \to s$) that are randomly selected
For iter = 1 to loopLimit
  While (tabuIter < tabuLoopLimit)
      bestVal = BIG
      For i = 1 to SizeOf(candList)
          $t_0 \to s_A$ = candList[i]
          If ($t_0 \to s_A$ is feasible)
             If (DelObjVal < bestVal)
                  If ($t_0 \to s_A$ is not tabu)
                     best_t = t0, best_s = $s_A$, best_$s_0$ = $s_0$
                  Else
                     If (DelObjVal + curObjVal < bestObjVal)
                     //override tabu status
                         best_t = $t_0$, best_s = $s_A$, best_$s_0$ = $s_0$
                     EndIf
                  EndIf
                  bestVal = DelObjVal
             EndIf
          EndIf
      EndFor
      If (bestVal < BIG)
          Insert an element(best_t, best_s)
```

```
            **Drop an element(best_t, best_s0)**
            curObjVal = DelObjVal + curObjVal
            If (curObjVal < bestObjVal)
               bestObjVal = curObjVal, tabuIter = 0
            EndIf
            **Updating move value** for impacted clusters and elements
         EndIf
         tabuIter ++
         candList = moves selected upon (2.27)
      EndWhile
      //diversification procedure
      Perform **Diversification** procedure describe above
      TabuEnd(t, s) = 0 for all elements t and clusters C(s)
   EndFor
```

loopLimit and *tabuLoopLimit* are set to be 1,000 and 100 respectively.

The overall tabu Search algorithm for solving cohesive cluster problems can then be summarized in the following simple form:

```
//Algorithm for solving cohesive clustering problems
Create initial solution
Tabu Search procedure
```

3 Computational experiment

We have implemented the algorithm of Sect. 2 using the C# programming language. The computational environment for all computational experiments mentioned below is a desktop with Windows 7 Enterprise operating system, CPU speed 2.7 GHz, and 4 GB of RAM.

Different datasets are used to validate the objective function listed in (2.10) which we repeat here for clarity:

$$ObjVal = min\left(\alpha_1 \sum\nolimits_1^{N_s} FV\left(s\right) + \alpha_2 \sum\nolimits_1^{N_s} VAR(s)\right) \tag{3.1}$$

An interesting finding emerged while investigating the impacts of parameters a_1 and a_2 on the overall solution quality. Figure 2 depicts the scenario where $\alpha_1 = 0.99$ and $\alpha_2 = 0.01$ while Fig. 3 displays the result when $\alpha_1 = 0.4$ and $\alpha_2 = 0.6$ (The *FV(s)* and *VAR(s)* values show in the figures are normalized. The X-axis identifies iterations while the Y-axis values are for $\sum_1^{N_s} FV\left(s\right)$ and $\sum_1^{N_s} VAR(s)$ respectively.)

The relationships shown are quite consistent for all datasets used for the computational experiments, disclosing that the introduction of variance in the objective function indeed helps to generate cohesive clusters. The use of variance appears to eliminate a lot of "nearly equal" move evaluations and makes it easier to pinpoint the

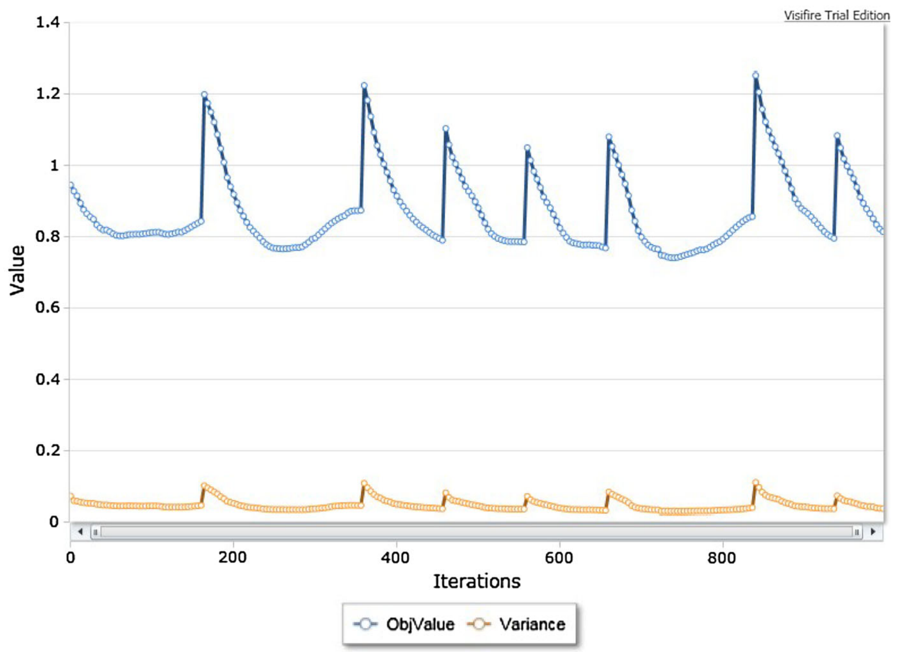


Fig. 2 Less consideration of variance

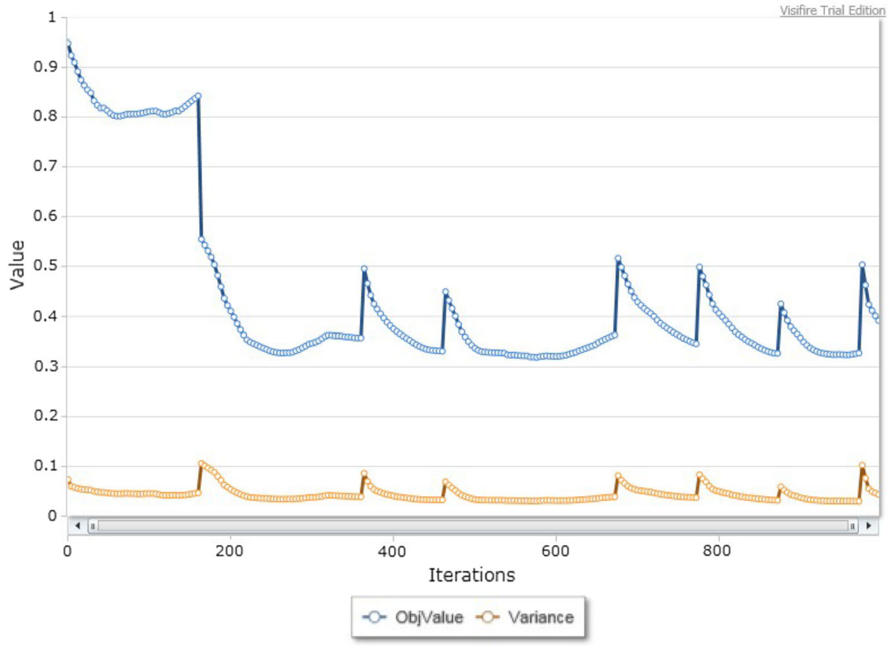


Fig. 3 Consideration of variance

truly relevant moves. During the computational experiment procedure, we divided the datasets into two parts: a trial set and a data application set. The trial set was used to tune the parameters α_1 and α_2 in the objective function. Based upon the experiments, it turned out that the results with $\alpha_1 = 0.4$ and $\alpha_2 = 0.6$ consistently outperformed the results obtained with other settings. We decided that this setting to be used for the data application dataset when the algorithm is compared to the well-known K-means. Accordingly, in the following computational experiments, α_1 is set to 0.4 and α_2 is set to 0.6 respectively, and these values don't vary from one run to the other.

In order to evaluate the algorithm more comprehensively we use two datasets to conduct the computational experiments:

- Experiment 1: data randomly generated and comparison to K-means' result.
- Experiment 2: data collected from a real cloud platform where data records need to be clustered or classified upon certain characteristics. The results are compared to the ones obtained by K-means.

K-means is the most popular clustering algorithm and is included in the open source package called *R*. In the following experiments we use the K-means algorithm in *R* to solve various clustering problems for the purposes of computational comparisons.

Experiment 1 In this computational experiment, we created the corresponding dataset by generating elements (points) randomly distributed on a plane. In order to determine and validate the capability of our algorithm to detect patterns embedded in the data, we also created a dataset that has certain patterns. The example of data with patterns is shown in the following picture:

The computational results of this dataset are shown in Table 1 where α_1 and $\alpha 2$ in objective function (2.10) are set to 0.4 and 0.6 respectively. We used different coordinate systems to generate different data sets containing data with patterns as well as data in which points are randomly distributed on a plane. Therefore the value scales listed in the table are different.

In the table we define:

- NP: number of points (elements) to be clustered
- NS: number of clusters
- CPT: compactness of clusters, i.e. $\sum_1^{N_s} FV(s)$
- SV: similarity value, i.e. $\sum_1^{N_s} VAR(s)$
- TS: our tabu search based clustering algorithm

From the computational results listed in Table 1, we are able to conclude that the computational times of K-means do not increase dramatically while those of our algorithm increase with the size of the problems to be solved, which is expected as our introduction of variance (to produce a refined guidance mechanism for achieving similarity and compactness) consumes additional time. Nevertheless in terms of solution quality, our algorithm beats K-means in both compactness and similarity of resultant clusters for all tested data instances except one. For the data instance that was the single exception, K-means is able to achieve better similarities of the elements within a cluster, but our algorithm is able to build more compact clusters.

Table 1 Results for dataset 1

Data type	NP	NS	Total time (mm:ss:ms)		CPT		SV	
			K-means	TS	K-means	TS	K-means	TS
Data without patterns	31	3	00:01:51	00:01:12	5.241	5.241	1.26	1.26
	100	3	00:01:51	00:01:23	238.29	219.81	4,443.88	2,460.81
	150	4	00:01:53	00:02:52	256.21	204.5	4,508.25	3,746.31
	300	3	00:01:59	00:04:67	153.87	153.28	3,473.55	3,414.28
	300	4	00:01:55	00:05:16	193.42	191.1	5,145.99	4,815.6
	350	7	00:01:55	00:10:91	357.41	334.79	4,224.7	3,591.15
	400	8	00:01:56	00:13:76	271.17	230.12	3,453.77	2,613.9
	400	9	00:01:56	00:12:45	334.24	251.35	3,591.43	3,787.1
	500	9	00:01:58	00:26:72	405.39	365.18	4,383.33	3,775.46
Data with patterns	300	3	00:01:54	00:05:13	35.31	35.09	95.22	92.99
	300	4	00:01:54	00:05:48	36.99	36.99	76.95	76.95
	300	5	00:01:63	00:09:32	31.57	30.69	64.07	61.96
	500	3	00:01:55	00:35:43	35.91	35.88	141.6	140.49
	500	4	00:01:54	00:30:12	26.12	26	46.76	46.36
	500	5	00:01:53	00:39:17	31.53	30.53	54.64	51.01

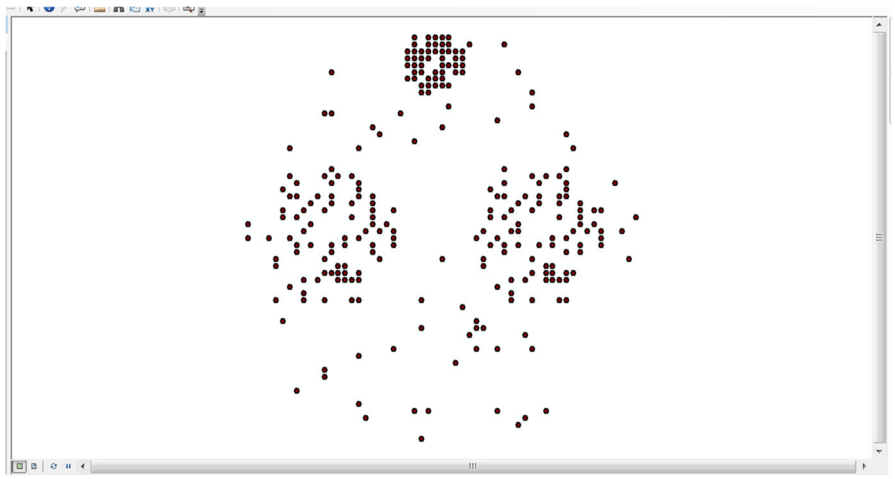


Fig. 4 Data with patterns

Based upon the computational results, we can also conclude that our algorithm has the capability of detecting the patterns embedded in a dataset. For the data shown in Fig. 4, the result depicted in the following picture (Fig. 5) confirms that our algorithm identifies the patterns properly in the dataset.

It is noted that the computational times for the datasets with patterns are bit longer. We realize that there is some room for improvement in the step that constructs initial clusters, and we plan to enhance this step in order to find better initial solutions. We

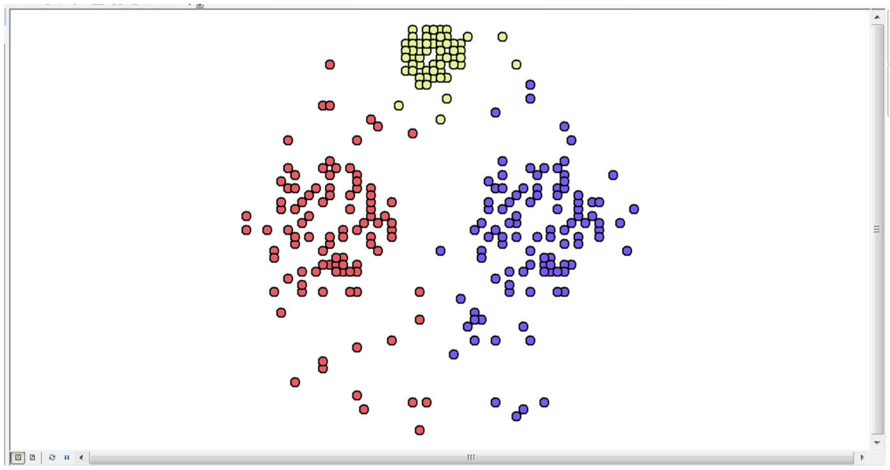


Fig. 5 Result of clustering dataset in Fig. 4

anticipate this will reduce the time spent in the improvement procedure to achieve solutions of high quality.

Experiment 2 This computational experiment employs the datasets (job data) collected from the cloud platform of one of the world's largest e-commerce companies, as noted in the Introduction. As we explained earlier the purpose of job clustering is to help create business rules to detect exceptions of certain jobs more effectively. Based upon the company analysis and business practice, a job can be characterized by: CPU times (in seconds), Memory required (in GB), and I/O size (in KB). The "distance" or score of any pair of jobs (elements) can be determined by applying the L2 Norm presented in (2.5). Of course all attributes will be normalized in order to reduce the data skew. The jobs involved in the computational experiment are selected randomly from the data set. The computational results for experiment 2 are presented in Table 2 (the symbols used this table are the same as those in Table 1).

Here it is obvious that our algorithm obtains much better solutions than K-means in terms of both the compactness of the resulting clusters and the similarity of their elements (jobs). While our computational times are longer than those of K-means, they are well within the acceptable range for our originally stated purpose of creating an algorithmic design in which our algorithm will be used off-line during a large part of its execution to conduct data preparation. Furthermore, our computational experiments also disclose that the jobs on the node where the experiment data was collected can be grouped into three or four major groups according to the evaluations provided by our similarity criteria. The relationships underlying this discovery have been verified to be more widely applicable by running additional tests on data outside of the training data used to build clusters in the tests reported above. Indeed our algorithm plays an important role for the company in preparing data and creating business rules to detect abnormal jobs associated with a node.

Table 2 Results for dataset 2

NP	NS	Total time (mm:ss:ms)		CPT		SV	
		K-means	TS	K-means	TS	K-means	TS
100	3	00:01: 54	00:01:38	2.28	0.84	2.74	1.26
100	4	00:01:52	00:01:81	2.37	0.77	2.74	1.23
200	3	00:01:55	00:02:87	8.38	1.05	26.52	8.65
200	4	00:01:56	00:03:19	8.48	1.12	26.55	9.39
200	8	00:01:54	00:04:39	8.55	1.57	26.57	12.87
300	3	00:01:57	00:04:93	0.79	0.28	0.19	0.12
350	4	00:01:81	00:09:54	5.19	0.93	14.87	4.85
400	4	00:01:58	00:10:54	7.53	0.62	12.74	4.27
500	3	00:01:64	00:17:97	5.11	0.89	12.96	4.54
500	5	00:01:61	00:17:26	6.04	0.93	13.06	4.76
500	10	00:01:56	00:19:89	6.39	0.93	13.11	4.76
800	3	00:01:74	01:32:33	6.11	0.89	12.45	5.39
800	4	00:01:77	01:33:12	24.02	1.58	139.51	15.54
1000	8	00:01:74	03:35:94	6.72	0.96	12.45	5.31

4 Conclusions

In this paper we propose a mathematical model that simultaneously accounts for compactness and similarity in creating clusters, and present a tabu search based algorithm to solve the corresponding formulation. Our new algorithm considers the compactness and similarity of clusters effectively, as demonstrated by computational tests on randomly generated datasets and also on datasets drawn from real world applications. The parameterized variance component introduced in the objective function proves effective for generating higher quality solutions. The elite candidate strategy employed in the tabu search improvement procedure has cut computational time significantly while providing good quality solutions. Our computational outcomes further disclose the algorithm's robustness, which enables a user to apply the algorithm without extensive tuning and without having to change parameter values to solve problems. These features afford significant advantages in preprocessing the datasets and creating business rules for detecting abnormally of a cloud platform in the practice.

Future research will focus on enhancements in order to solve problems more efficiently, including (1) enhancing the initial solution step to detect patterns more quickly; (2) employing a more advanced elite candidate strategy to reduce computational time further; (3) utilizing a multi-core CPU to speed up the algorithm. The successful outcomes of our current algorithmic design give a useful foundation for pursuing such enhancements.

Acknowledgments The authors would like to thank our student team including Aro Lee, Zheng Xu, and Jiayao Gao for their efforts in implementing the algorithm, data preparations, and partial computational experiments. We would also like to express our gratitude to two anonymous referees for their valuable criticisms and suggestions to improve our manuscript. This research is partially supported by project contract CIUC20140004.

References

April, J., Better, M., Glover, F., Kelly, J.P., Kochenberger, G.: Strategic workforce optimization: ensuring workforce readiness with OptForce. Ann. Optim. (2014, in press)

Bae, E., Bailey, J., Dong, G.: A clustering comparison measure using density profiles and its application to the discovery of alternate clusterings. Data Min. Knowl. Discov. **21**, 427–477 (2010)

Brusco, M.J., Steinley, D., Cradit, J.D., Singh, R.: Emergent clustering methods for empirical OM research. J. Oper. Manag. **30**, 454–466 (2012)

Cao, B., Glover, F.: Creating balanced and connected clusters for improved service delivery routes in logistics planning. J. Syst. Sci. Syst. Eng. **19**, 453–480 (2010)

Datta, S., Giannella, C.R., Kargupta, H.: Approximate distributed K-means clustering over a peer-to-peer network. IEEE Trans. Knowl. Data Eng. **21**(10), 1372–1388 (2009)

Glover, F.: Future paths for integer programming and links to artificial intelligence. Comput. Oper. Res. **13**, 533–549 (1986)

Glover, F., Laguna, M.: Tabu Search. Kluwer Academic Publishers, Dordrecht (1997)

Glover, F., Laguna, M.: Tabu search: effective strategies for hard problems in analytics and computational science. In: Pardalos, P.M., Du, D.-Z., Graham, R.L. (eds.) Handbook of Combinatorial Optimization, 2nd ed, vol. XXI, pp. 3261–3362. Springer, New York (2013)

Grabmeier, J., Rudolph, A.: Techniques of cluster algorithms in data mining. Data Min. Knowl. Discov. **6**, 303–360 (2002)

Kochenberger, Glover, F., Alidaee, B., Wang, H.: Clustering of microarray data via clique partitioning. J. Comb. Optim. **10**, 77–92 (2005)

Kochenberger, G.A., Hao, J.K., Lü, Z., Wang, H., Glover, F.: Solving large scale Max Cut problems via tabu search. J. Heuristics **19**(4), 565–571 (2013)

Linoff, G.S., Berry, M.J.: Data Mining Techniques, 3rd edn. Wiley Publishing Inc, Indianapolis, IN (2011)

Liu, C.M.: Clustering techniques for stock location and order-picking in a distribution center. Comput. Oper. Res. **26**, 989–1002 (1999)

Provost, F., Fawcett, T.: Data Science for Business. O'Reilly Media, Inc., Sebastopol, CA (2013)

Strehl, A., Ghosh, J.: Relationship-based clustering and visualization for high-dimensional data mining. INFORMS J. Comput. **15**, 1–23 (2002)

Trappey, C.V., Trappey, A.J.C., Chang, A.C., Huang, A.Y.L.: Clustering analysis prioritization of automobile logistics services. Ind. Manag. Data Syst. **110**(5), 731–743 (2010)

Wu, Q., Hao, J.K., Glover, F.: Multi-neighborhood tabu search for the maximum weight clique problem. Ann. Oper. Res. **196**(1), 611–634 (2013)

Wu, H., Wang, X., Peng, Z., Li, Q.: Div-clustering: exploring active users for social collaborative recommendation. J. Netw. Comput. Appl. **36**(6), 1642–1650 (2013)