Constraint-Based Clustering in Large Databases

Anthony K. H. Tung[1], Jiawei Han[1], Laks V.S. Lakshmanan[2], and Raymond T. Ng[3]

[1] Simon Fraser University, {khtung, han}@cs.sfu.ca
[2] IIT – Bombay & Concordia U, laks@cs.concordia.ca
[3] University of British Columbia, rng@cs.ubc.ca

Abstract. Constrained clustering—finding clusters that satisfy user-specified constraints—is highly desirable in many applications. In this paper, we introduce the constrained clustering problem and show that traditional clustering algorithms (e.g., k-means) cannot handle it. A scalable constraint-clustering algorithm is developed in this study which starts by finding an initial solution that satisfies user-specified constraints and then refines the solution by performing confined object movements under constraints. Our algorithm consists of two phases: *pivot movement* and *deadlock resolution*. For both phases, we show that finding the optimal solution is NP-hard. We then propose several heuristics and show how our algorithm can scale up for large data sets using the heuristic of *micro-cluster sharing*. By experiments, we show the effectiveness and efficiency of the heuristics.

1 Introduction

Cluster analysis has been an active area of research in computational statistics and data mining with many algorithms developed. However, few algorithms incorporate user-specific constraints in cluster analysis. Many studies show that constraint-based mining is highly desirable since it often leads to effective and fruitful data mining by capturing application semantics [NLHP98,KPR98,LNHP99]. This is also the case in cluster analysis.

Formally, the unconstrained clustering problem can be defined as follows.
Unconstrained Clustering (UC): Given a data set D with n objects, a distance function $df : D \times D \longrightarrow \Re$, and a positive integer k, find a *k-clustering*, i.e., a partition of D into k disjoint clusters $(Cl_1, \ldots, Cl_k)$ such that $DISP = (\sum_{i=1}^{k} disp(Cl_i, rep_i))$ is minimized.

The "dispersion" of cluster Cl_i, $disp(Cl_i, rep_i)$, measures the total distance between each object in Cl_i and the *representative* rep_i of Cl_i, i.e., $disp(Cl_i, rep_i)$ is defined as $\sum_{p \in Cl_i} df(p, rep_i)$. The representative of a cluster Cl_i is chosen such that $disp(Cl_i, rep_i)$ is minimized. Finding such a representative for each cluster is generally not difficult. For example, the k-means algorithm uses the centroid of the cluster as its representative, which can be calculated in linear time.

The constrained clustering problem can be defined as follows.

Constrained Clustering (CC): Given a data set D with n objects, a distance function $df : D \times D \longrightarrow \Re$, a positive integer k, and *a set of constraints* $\mathcal{C}$, find a *k-clustering* $(Cl_1, \ldots, Cl_k)$ such that $DISP = (\sum_{i=1}^{k} disp(Cl_i, rep_i))$ is minimized, and *each cluster Cl_i satisfies the constraints $\mathcal{C}$*, denoted as $Cl_i \models \mathcal{C}$.

A fundamental difference between the UC and CC problems is that the unconstrained clustering algorithms are designed to find clusterings satisfying the *nearest rep(resentative) property* (NRP), defined below, whereas for the CC problem, the NRP may conflict with constraint satisfaction.

The Nearest Rep(resentative) Property (NRP): Let $(Cl_1, ..., Cl_k)$ be the k-clustering computed by the algorithm, and let rep_i denote the representative of cluster Cl_i, $1 \le i \le k$. Then a data object $p \in D$ is placed in a cluster Cl_j iff rep_j is the closest to p among all the representatives, i.e., $(\forall p \in D)(\forall 1 \le j \le k)\ [p \in Cl_j \ \Leftrightarrow\ (\forall i \ne j)\ df(p, rep_j) \le df(p, rep_i)]$.

In this paper, we study the CC problem. A taxonomy of constraints useful in applications is presented in Section 2. In Section 3, we review works related to the CC problem, and in Section 4, we analyze the major challenges of CC. In Section 5, we develop an algorithm for CC under an existential constraint. In Section 6, we study how to scale up our algorithm by *micro-cluster sharing*. The experiments evaluating the effectiveness of the proposed heuristics are reported in Section 7. Section 8 discusses the handling of other SQL aggregate constraints, and Section 9 concludes the paper.

2 A Taxonomy of Constraints for Clustering

Depending on the nature of the constraints and applications, the CC problem can be classified into the following categories.

1. Constraint on individual objects: This constraint confines the set of objects to be clustered, e.g., cluster only luxury mansions of value over one million dollars. It can be easily handled by preprocessing (e.g., performing selection using an SQL query), after which the problem reduces to an instance of the UC problem.
2. Obstacle objects as constraints: A city may have rivers, bridges, highways, lakes, mountains, etc. Such obstacles and their effects can be captured by redefining the distance functions $df()$ among objects. Once that is done, the problem again reduces to an instance of the UC problem.
3. Clustering parameters as "constraints": Some "constraints" may serve as the parameters in a clustering algorithm, e.g., the number of clusters, k. Such parameters, though specifiable by users, are not considered as constraints in our study.
4. Constraints imposed on each individual cluster: This is the theme of our study. Within this class, we focus on constraints formulated with SQL aggregates.

Let each object O_i in the database D be associated with a set of m attributes $\{A_1, \ldots, A_m\}$. The value of an attribute A_j of an object O_i is denoted as $O_i[A_j]$.

Definition 1 (SQL Aggregate Constraints). Consider the aggregate functions $agg \in \{max(), min(), avg(), sum()\}$. Let θ be a comparator function, i.e., $\theta \in \{<, \leq, \neq, =, \geq, >\}$, and c represent a numeric constant. Given a cluster Cl, an SQL aggregate constraint on Cl is a constraint in one of the following forms: (i) $agg(\{O_i[A_j] \mid O_i \in Cl\})\ \theta\ c$; or (ii) $count(Cl)\ \theta\ c$. □

In this paper, we mainly focus on one type of constraints, called **existential constraints**:

Definition 2 (Existential Constraints). Let $W \subset D$ be any subset of objects. We call them **pivot** objects. Let c be a positive integer. An **existential constraint** on a cluster Cl is a constraint of the form: $count(\{O_i | O_i \in Cl, O_i \in W\}) \geq c$. □

Pivot objects are typically specified via constraints or other predicates. For example, in a market segmentation problem, pivot objects might be *frequent customers*. See Section 8 for a discussion on the generality of existential constraints.

3 Related Work

Cluster analysis has been an active area in computational statistics and data mining. Clustering methods can be categorized into partitioning methods [KR90,NH94], hierarchical methods [KR90,ZRL96], density-based methods [EKSX96], grid-based methods [WYM97,AGGR98], and model-based methods [HaKa00]. However, none of the existing methods incorporates user-specified constraints.

A problem somewhat similar to the CC problem is the facility location problem [STA97], mostly studied in operational research and theoretical computer science. It tries to locate k facilities to serve n customers such that the traveling distance from the customers to their facility is minimized. However, the only type of constraints they studied are constraints on the capacity of the facility, i.e., each facility can only serve a limited number of customers. If we assume that customers cannot be "split" between two facilities (as we do for CC), the resultant solution will require an increase both in the number of facilities and in the capacity of these facilities. However, if the customers can be "split", only the number of facilities needs to be increased. Such an increase in number of facilities and capacity is inappropriate for the CC problem as we treat user's constraints as hard constraints.

Since CC is a kind of constrained optimization problem, mathematical programming naturally comes to mind. Our concern, however, is its scalability with respect to a large database. To cluster n customers into k clusters, a mathematical programming approach will involve at least $k \times n$ variables. As n can be as large as a few millions, it is very expensive to perform mathematical programming. As can be seen later, our solution for handling a very large dataset involves a novel concept called *micro-cluster sharing*. This may correspond to dynamic combining and splitting of equations in a mathematical program, which has not been considered in mathematical programming but could be an interesting future direction for performing mathematical programming in large databases.

4 The Nearest Representative Property (NRP)

We first consider the theoretical implication of adding constraints to clustering, by examining the popular k-means algorithm although the discussion generalizes to other algorithms, such as the k-medoids algorithm.

Given a set of constraints $\mathcal{C}$, a "solution" space for the CC problem is defined as,

$$ClSp(\mathcal{C}, k, D) = \{(Cl_1, \ldots, Cl_k) \mid \forall 1 \leq i, j \leq k : \emptyset \subset Cl_j \subset D \;\&\; Cl_j \models \mathcal{C} \;\& \\ \cup\, Cl_j = D \;\&\; Cl_i \cap Cl_j = \emptyset, \text{ for } i \neq j\}$$

We refer to $ClSp(\mathcal{C}, k, D)$ as the (constrained) clustering space. Clusterings found by the k-means algorithm satisfy the NRP. Accordingly, the *constrained mean solution space* is defined as:

$$MeanSp(\mathcal{C}, k, D) = \{(Cl_1, \ldots, Cl_k) \mid (Cl_1, \ldots, Cl_k) \in ClSp(\mathcal{C}, k, D) \\ \& \; \forall 1 \leq j \leq k, \forall q \in D : (q \in Cl_j \;\Leftrightarrow\; (\forall i \neq j : df(q, p_j) \leq df(q, p_i)))\}$$

where p_j is the centroid of cluster Cl_j. It should be clear by definition that the mean space $MeanSp()$ is a strict subset of the clustering space $ClSp()$. To understand the role played by the NRP, let us revisit the situation when the set of constraints $\mathcal{C}$ is empty. The k-means algorithm does the smart thing by operating in the smaller $MeanSp()$ space than in the $ClSp()$ space. More importantly, the following theorem says that there is no loss of quality. Unless stated otherwise, proofs of the results in this paper can be found in [TNLH00] but are omitted here for lack of space.

Theorem 1. A clustering $\mathcal{UCL}$ is an optimal solution to the UC problem in the space $ClSp(\emptyset, k, D)$ iff it is an optimal solution to the UC problem in the mean space $MeanSp(\emptyset, k, D)$. □

Like virtually all existing clustering algorithms, the k-means algorithm does not attempt to find the global optimum. This is because the decision problem corresponding to k-clustering is NP-complete even for $k = 2$ [GJ79]. Thus, the k-means algorithm focuses on finding local optima. Theorem 1 can be generalized from the global optimum to a local optimum.

The point here is that $MeanSp(\emptyset, k, D)$ contains the "cream" of $ClSp(\emptyset, k, D)$, in that the global and local optima in $ClSp(\emptyset, k, D)$ are also contained in the smaller $MeanSp(\emptyset, k, D)$. This nice situation, however, does not generalize to the CC problem. For example, suppose there are only four customers with three located close to each other at one end of a highway, and the fourth at the other end. If the CC problem is to find two clusters with (at least) two customers in each, it is easy to see that it is impossible to satisfy the constraint and the NRP simultaneously.

To resolve this conflict, we adopt the policy that the user-defined constraints take precedence over the NRP. Specifically, the algorithm to be presented next regards the set $\mathcal{C}$ to be hard constraints that must be satisfied. The NRP, on the other hand, is treated as a "soft" constraint in the sense that it is satisfied as much as possible by the minimization of $(\sum_{i=1}^{k} disp(Cl_i, rep_i))$. But there is no guarantee that every object is in the cluster corresponding to its nearest center.

5 Clustering without the Nearest Representative Property

In this section, we will develop an algorithm to perform CC under an existential constraint. An important difference of our method from the UC algorithms is that our algorithm tries to find a good solution by performing cluster refinement in the constraint space, $ClSp(\mathcal{C}, k, D)$, which we represent using a *clustering locality graph*, $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, described as follows:

- The set $\mathcal{V}$ of nodes is the set of all k-clusterings. More precisely, it is the unconstrained clustering space $ClSp(\emptyset, k, D)$. Nodes which satisfy existential constraint (EC) are called *valid nodes*, and those that do not are called *invalid nodes*.
- There is an edge e between two nodes $\mathcal{CL}_1, \mathcal{CL}_2$ in the graph iff they are different by only one pivot object, i.e., $\mathcal{CL}_1$ of the form $(Cl_1, \ldots, Cl_i, \ldots, Cl_j, \ldots, Cl_k)$, whereas $\mathcal{CL}_2$ of the form $(Cl_1, \ldots, Cl_i - \{p\}, \ldots, Cl_j \cup \{p\}, \ldots, Cl_k)$ for some pivot object $p \in Cl_i$ & $j \neq i$. If a node $\mathcal{CL}_2$ is connected to $\mathcal{CL}_1$ by an edge, then $\mathcal{CL}_2$ is called a *neighbor* of $\mathcal{CL}_1$ and vice versa.

With such a graph, a naive algorithm to solve the CC problem given k and EC is to first pick a valid node in the locality graph and move to a valid neighboring node which gives the highest decrease in $DISP$. Intuitively, such a node movement is a cluster refinement process similar to the k-means algorithm which tries to refine the clustering by moving objects to the nearest center to reduce $DISP$. The cluster refinement process terminates when no node of lower $DISP$ is found. The algorithm will then output $\mathcal{CL}$ as the solution. However, this is a generate-and-test algorithm which is inefficient since the number of neighbors of a node is potentially large. To improve its efficiency, the number of nodes to be examined needs to be restricted.

5.1 Cluster Refinement Under Constraints

To derive a more efficient algorithm for CC, we first define a set of *unstable pivots* given a valid node $\mathcal{CL} = (Cl_1, \ldots Cl_k)$.

Definition 3. (Unstable Pivots) A set of unstable pivots, S, with respect to $\mathcal{CL}$ is a collection of all pivots in D such that each $s \in S$ belongs to some Cl_i in $\mathcal{CL}$ but s is nearer to a representative of some Cl_j, $j \neq i$. □

Using S, we form a subgraph of $\mathcal{G}$, viz., $\mathcal{SG} = (\mathcal{SV}, \mathcal{SE})$, where the set of nodes $\mathcal{SV}$ is defined as follows: (1) (base case) the initial node $\mathcal{CL}$, representing the chosen valid clustering is in $\mathcal{SV}$; (2) (inductive case) for any node $\mathcal{CL}$ in $\mathcal{SV}$, *if* (i) there is an object s in Cl_i whose nearest cluster representative is in Cl_j, and (ii) $\mathcal{CL}$ is of the form $(Cl_1, \ldots, Cl_i, \ldots, Cl_k)$, *then* the node $\mathcal{CL}'$ of the form $(Cl_1, \ldots, Cl_i - \{s\}, \ldots, Cl_j \cup \{s\}, \ldots, Cl_k)$ is also in $\mathcal{SV}$; and (3) there is no other node in $\mathcal{SV}$. Intuitively, once S is defined, the subgraph $\mathcal{SG}$ includes all the nodes that are reachable from $\mathcal{CL}$ via the movements of some $s \in S$ to their nearest cluster. Let us denote the $DISP$ of any node v with respect to a set of representatives REP as $DISP_{REP}(v)$.

Theorem 2. *$DISP_{REP'}(\mathcal{CL}') \leq DISP_{REP}(\mathcal{CL})$ for any nodes $\mathcal{CL}, \mathcal{CL}'$ in $\mathcal{SG}$ as above, REP and REP' being the set of representatives for $\mathcal{CL}$ and $\mathcal{CL}'$ respectively.*
Proof: Let $REP = (rep_1, ..., rep_k)$ and $REP' = (rep'_1, ..., rep'_k)$. The dispersion of $\mathcal{CL}'$ calculated *with respect to REP* will be $DISP_{REP}(\mathcal{CL}') = (\sum_{i=1}^{k} disp(Cl'_i, rep_i))$. We first observe that

$$DISP_{REP}(\mathcal{CL}') \leq DISP_{REP}(\mathcal{CL})$$

This is because the set of representatives is the same on both sides of the inequality and since $\mathcal{CL}'$ can be obtained by moving some $s \in S$ to their nearest representative in REP, the reduction in dispersion will result in the above observation. On the other hand, since REP' is a set of representatives for $\mathcal{CL}'$, by definition they will minimize the dispersion for Cl'_1,...,Cl'_k, we thus have the following inequality,

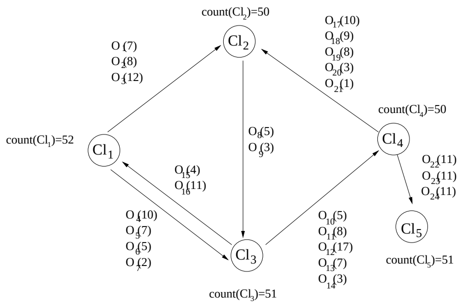
$$DISP_{REP'}(\mathcal{CL}') \leq DISP_{REP}(\mathcal{CL}')$$

By combining these two inequalities together, we have

$$DISP_{REP'}(\mathcal{CL}') \leq DISP_{REP}(\mathcal{CL}') \leq DISP_{REP}(\mathcal{CL}) \qquad \square$$

By Theorem 2, we conclude that our clusters can in fact be refined just by searching $\mathcal{SG}$. There are two advantages to doing this. First, our efficiency improves because the number of nodes to be searched is reduced, and the movement always leads to progressive refinement in clustering quality. This in itself does *not* guarantee the chosen neighbor is valid. Second, instead of considering only neighbors, $\mathcal{SG}$ allows us to consider nodes that are many steps away.

Given $\mathcal{SG}$, we adopt the steepest descent approach and plan a path along the valid nodes of $\mathcal{SG}$ which leads to a new valid node $\mathcal{CL}'$ with minimized dispersion in $\mathcal{SG}$. We call this problem the *Best Path (BP) Problem.* To plan the path, only unstable pivots in a surplus cluster (cluster which have more objects than required by EC) can be moved. We call such an object, a *movable* object. To gain more insight into the BP problem and to derive an algorithm for solving it, we introduce a concept called *pivot movement graph* which can be used to represent the state of clustering in each node of $\mathcal{SG}$.



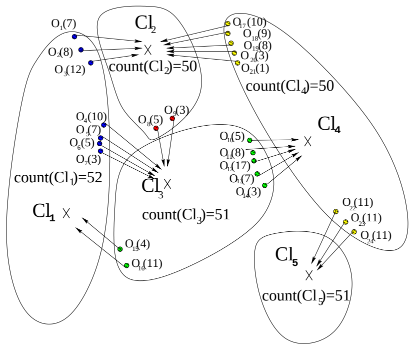


Fig. 1. A Pivot Movement Graph **Fig. 2.** The Actual Situation.

Definition 4. *(Pivot Movement Graph)* A *pivot movement graph* is a directed graph in which each cluster is represented by a node. An edge from Cl_i to Cl_j indicates that there is at least one unstable pivot object in Cl_i that has Cl_j as its nearest center. These objects are represented as labels on the edge. The reduction in $DISP$ when an unstable object is moved to its nearest center is shown next to each of these objects. □

Figure 1 shows an example of a pivot movement graph which is under the constraint "$\forall i, count(Cl_i) \geq 50$". As such, the surplus clusters at this instance are Cl_1, Cl_3 and Cl_5. Figure 2 shows the actual situation depicted by the pivot movement graph in Figure 1. For clarity, only the unstable pivots and the cluster representatives (marked by a "$\times$") are shown. Given a pivot movement graph, a *Pivot Movement (PM)* problem is the problem of computing a schedule of movements for the unstable objects in the graph such that the total reduction in $DISP$ is maximized.

Theorem 3. *The BP problem is equivalent to the PM problem.*

Proof: Given an optimized solution for BP, we follow the path given in the solution and move the pivots in the corresponding pivot movement graph. This will give a maximized reduction in dispersion. Similarly, if an optimized schedule is given for PM, we can follow the schedule and move along a path where each node in the path corresponds to a state of the pivot movement graph when the schedule is followed. This will bring us to a node with minimized dispersion in $\mathcal{SG}$. □

Given their equivalence, it suffices to focus on the PM problem.

Definition 5. (The PM Decision Problem) Given a pivot movement graph and an existential constraint EC, the PM decision problem is to determine whether there is a schedule of movements of objects around the clusters such that EC is satisfied at all times and the total dispersion being reduced is $\geq B$ where B is a numeric constant. □

Two observations hint at the difficulty of this problem. (1) The movement of an unstable pivot object could possibly trigger a series of movements of other unstable pivot objects. For example, by moving O_3 from Cl_1 to Cl_2, Cl_2 now has 51 pivot objects, and thus we could move O_8 from Cl_2 to Cl_3. We refer to such a series of triggerings as a **movement path**. (2) Given a surplus cluster with more than one outgoing edge, the choice of the outgoing edge that minimizes $DISP$ in the resultant movement path is not obvious. Indeed we can show:

Theorem 4. The PM decision problem is NP-complete.
Proof: See [TNLH00]. $\square$

Furthermore, by using a result given in [KMR97], we can show that it is not possible to compute in polynomial time a constant factor approximation for the PM problem. Thus, an alternative is to use heuristics which could work well in practice and efficient enough for handling a large dataset. The purpose of the heuristic is to iteratively pick an edge in the pivot movement graph and move an unstable object on the edge to its nearest representative thus forming a schedule of movements for the unstable pivots.

We experiment with two heuristics. The first is a **random heuristic** in which a random edge is selected from those edges that originate from a surplus cluster; whereas the second is a **look-ahead** l heuristic which looks ahead at all possible movement paths originating from a surplus cluster, of length up to l, and selects the best among them. The selected movement path is then activated, resulting in a movement of up to l objects depending on the length of the path. Since there are at most $k(k-1)$ edges, there are at most $O(k(k-1)^{l+1})$ movement paths. While there exist optimization strategies that can avoid examining all the qualifying movement paths of length l, the worst case complexity of this heuristic remains $O(k(k-1)^{l+1})$. Thus, the value of l is designed to be a small integer.

Using these heuristics, our corresponding movement in $\mathcal{SG}$ will eventually reach a node $\mathcal{CL}''$ where future movement is no more possible. We then repeat the process and form a new subgraph $\mathcal{SG}$ for processing.

5.2 Handling Tight Existential Constraints

While the cluster refinement algorithm discussed earlier works well under most constraints, problem arises when the constraint EC is tight, i.e., when it is nearly impossible to be satisfied. For example, given $k = 5$, $|D| = 100$ and $EC = \{count(Cl_i) \geq 20\}, 1 \leq i \leq 5$, our algorithm may get into a *deadlock cycle*. A sequence of clusters $\langle Cl_1, \ldots, Cl_k, Cl_1 \rangle$ is said to be in a *deadlock cycle* of length k if (a) all the clusters are non-surplus; and (b) there is an edge in the pivot movement graph from Cl_i to $Cl_{i+1}, 1 \leq i \leq k-1$ and one from Cl_k to Cl_1, respectively.

In terms of the graph, $\mathcal{SG}$, a tight EC means that $\mathcal{SG}$ contains a large number of invalid nodes and refining the clusters by movement through only valid nodes is not possible. In view of this, a deadlock resolution phase is added before computing a new subgraph $\mathcal{SG}$. The objective of the deadlock resolution phase is to provide a mechanism to jump over a set of invalid nodes by resolving deadlock in the pivot movement graph. Similar to the PM problem, we can prove (in a way similar to that for Theorem 4) that resolving deadlock optimally is NP hard.

Similarly, we can show that there is also no constant factor approximation algorithm for the deadlock resolution problem which runs in polynomial time. Thus, we resort to the following heuristic based on a randomized strategy. It conducts a depth-first search on the pivot movement graph to find any deadlock cycle. Suppose the deadlock cycle detected is $\langle Cl_1, \ldots, Cl_k, Cl_1 \rangle$. Let n_i denote the number of unstable pivot objects appearing as labels on the edge from Cl_i to Cl_{i+1}. Then let n_{max} denote the minimum n_i value among the edges in the cycle, i.e., $n_{max} = min_{1 \leq i \leq k}\{n_i\}$. This marks the *maximum* number of unstable objects that can be moved across the *entire* cycle without violating EC. Once the n_{max} value has been determined, the heuristic would move the unstable pivot objects with the top-n_{max} highest reduction in $DISP$ across each edge of the cycle causing the cycle to be broken.

5.3 Local Optimality and Termination

Having introduced our algorithm, we will now look at its formal properties by analyzing the two main phases of the algorithm: *pivot movement* and *deadlock resolution*. Our algorithm essentially iterates through these two phases and computes a new subgraph $\mathcal{SG}$ at the end of each iteration.

Local Optimality Result. Having modeled our cluster refinement algorithm as a graph search, we would like to establish that at the end of each iteration, the clustering obtained corresponds to a local minimum in the subgraph $\mathcal{SG}$. However, since all dispersion of nodes in $\mathcal{SG}$ is actually computed with respect to the cluster representatives of $\mathcal{CL}$, when there is a pivot movement, say object p moved from Cl_i to Cl_j, both the representatives of Cl_i and that of Cl_j change, and the set of unstable pivots S can also change, which means that $\mathcal{SG}$ itself must be recomputed. This process is time-consuming, especially for our look-ahead heuristic which must recompute $\mathcal{SG}$ every step it looks ahead. Because of this, we choose to freeze the representatives of each cluster and avoid the recomputation of $\mathcal{SG}$. As such, the cost of each node $\mathcal{CL}$ in the subgraph $\mathcal{SG}$ is not the true dispersion but rather the "approximated" dispersion, denoted as $\widehat{disp}(\mathcal{CL})$, relative to the fixed representatives. Now we can establish the following result. Intuitively, at the end of the pivot movement phase, no surplus cluster in the pivot movement graph has an outgoing edge. Thus, it is not possible to find a valid neighbor of the current one that has a lower dispersion.

Lemma 1. The clustering obtained at the end of the pivot movement phase is a local minimum in the subgraph $\mathcal{SG}$, where cost is based on approximated dispersion $\widehat{disp}(\mathcal{CL})$. $\square$

Interestingly, a deadlock cycle of length k corresponds to a path $\langle \mathcal{CL}_1, \ldots, \mathcal{CL}_{k+1} \rangle$ in $\mathcal{SG}$, such that the first node/clustering $\mathcal{CL}_1$ and the last node $\mathcal{CL}_{k+1}$ are valid, but *all the other nodes are not*. This is a very interesting phenomenon because resolving a deadlock cycle amounts to jumping from one valid node to another via a sequence of invalid nodes in $\mathcal{SG}$. In particular, if deadlock cycles are resolved after the pivot movement phase as in our algorithm, then we jump from a valid local minimum to another (which is not a neighbor) with a strictly lower value of dispersion.

Lemma 2. The clustering obtained at the end of the deadlock resolution phase is a local minimum in the subgraph $\mathcal{SG}$, where cost is based on approximated dispersion $\widehat{disp}(\mathcal{CL})$. □

Termination of the Algorithm. Since each move in the graph $\mathcal{SG}$ corresponds to a reduction in the number of unstable pivot objects, and the number of unstable pivot objects is finite, both the object movement phase and deadlock resolution phase will terminate. Moreover, since we move to a node of lower $DISP$ for every iteration, and $\mathcal{G}$ is a finite clustering space, it is impossible to have the $DISP$ value decreasing forever. Thus, the algorithm terminates.

6 Scaling the Algorithm by Micro-Cluster Sharing

For clustering large, disk-resident databases, many studies have adopted a *micro-clustering* methodology (e.g., [ZRL96,WYM97,BFR98,KHK99]), which "compresses" data objects into *micro-clusters* in a pre-clustering phase so that the subsequent clustering activities can be accomplished at the micro-cluster level. To ensure that not much quality is lost, a maximum radius on a micro-cluster is imposed.

By micro-clustering, in our cluster refinement, instead of moving one unstable object across the edges of a pivot movement graph at a time, we have to move one micro-cluster. However, since each micro-cluster can contain more than one pivot object, it may not be possible to move a micro-cluster away from a surplus cluster without invalidating the constraint. Similar complication arises when resolving deadlock since there is no guarantee that for each edge in a cycle, the total number of pivot objects in the micro-clusters to be moved add up to exactly n_{max}.

To resolve these problems, we introduce a novel concept called *micro-cluster sharing*. Given a micro-cluster with n non-pivot objects and m pivot objects, the n non-pivot objects will always be allocated to the nearest cluster, while the m pivot objects can be shared among multiple clusters. For example, consider Figure 3 in which micro-cluster mc_1 is formed from 5 non-pivot objects and 6 pivot objects. It is shared by three clusters, Cl_1, Cl_2 and Cl_3. Since Cl_2 is the nearest to mc_1, it owns all 5 of mc_1's non-pivot objects and also 2 pivot objects from mc_1. Cl_1, on the other hand, contains 3 pivot objects from mc_1, while Cl_3 has 1 pivot object from mc_1.

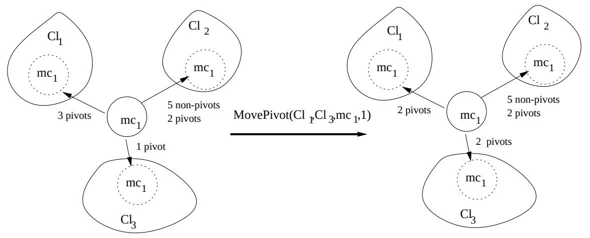


Fig. 3. An Example of Micro-cluster Sharing

To record the sharing or "splitting" of mc_1 into multiple clusters, we use the notation $Cl_i.mc_1$ to represent the part of mc_1 that is in Cl_i. During the pivot movement and deadlock resolution phases, if p objects of $Cl_i.mc_1$ are to be moved to Cl_j, the algorithm calls a function MovePivot(Cl_i, Cl_j, mc_1, p) which updates the numbers in $Cl_i.mc_1$ and $Cl_j.mc_1$ accordingly. In Figure 3, MovePivot(Cl_1, Cl_3, mc_1, 1) moves one pivot object from $Cl_1.mc_1$ to $Cl_3.mc_1$.

Given the MovePivot() function, the problem of being unable to shift micro-clusters around the clusters is effectively solved since the micro-clusters can now be dynamically split and combined to cater to the condition for swapping. Since the number of objects in a micro-cluster is small enough for all of them to fit in main memory, the above heuristic requires a minimum amount of I/O.

The remaining issue that we need to address is at the end of clustering, how to determine the *actual objects* in a micro-cluster mc that are to be assigned to $Cl_1, \ldots, Cl_q$, where these are all the clusters for which $Cl_i.mc$ is positive. We adopt the following greedy heuristic: For all the non-pivot objects in mc, they are all assigned to the nearest center/cluster. This is to reduce $DISP$ as much as possible. Consider the set of distances defined as: $\{df(O, Cl_i) \mid O$ is a pivot object in mc, and $1 \leq i \leq q\}$. Sort this set of distances in ascending order. Based on this order, the pivot objects are assigned to the cluster as near as possible, while satisfying the numbers recorded in $Cl_1.mc, \ldots, Cl_q.mc$.

7 Performance Analysis

We report our performance study, which evaluates the efficiency and effectiveness of the proposed heuristics. All the experiments were performed on a 450Mhz Intel Celeron, with 64MB of main memory, and an IBM 7200 rpm disk-drive.

Two datasets were used in our experiments. The first, DS1, is a dataset of a courier company for planning collection centers based on the locations of their frequent customers (see [TNLH00]). The second, DS2, is synthetic, generated following the synthetic datasets used in [ZRL96]. For lack of space, we report our experiments on DS2 only.

For the constraints, we made all data objects to be pivot objects in order to give most vigorous tests to our algorithms. Micro-clustering in our experiments was done using the CF-tree in the BIRCH algorithm [ZRL96] which only needs to scan through the database once. Note that BIRCH is used here as a pre-processing step and it's data structure is not utilized in any part of our algorithm.

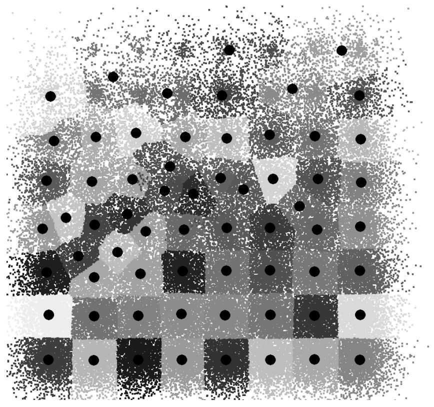
To separate the different heuristics used in our algorithms, we denote an algorithm as **RandLS** if it uses the *random heuristic* in pivot movement and **LAHLS-l** if *look-ahead-l heuristic*. For the micro-cluster sharing version of the two algorithms, the term "Micro" is added, i.e., **MicroRandLS** and **MicroLAHLS-l**.

Our synthetic dataset was generated using a modification of the synthetic dataset from [ZRL96], with skewed density distribution for testing the scalability of our algorithms and how constraints affect the clustering of a dataset. Essentially, there was a $M \times M$ grid in which cluster centers were placed. The distance between neighboring centers in the same row or column was set to 1. For a cluster centered at the coordinate $(row, column)$, $((row - 1) \times M + column) \times 50$ points were generated following a 2-d normal distribution with center at $(row, column)$ and variance 0.5^2. By design, the density of the synthetic data was skewed, being least dense at the top-left corner of the grid and most dense at the bottom-right. The value of M was varied from 4 to 8, generating datasets with various sizes and numbers of clusters. Figure 5 shows the parameters used for each dataset.

As shown in Figure 6(a), the order of effectiveness of the various algorithms generally remains unchanged with *LAHLS* giving the best quality of clustering. The running times of both *MicroRandLS* and *MicroLAHLS-4* remain relatively low as both $|D|$ and k increase. For a dataset with 104000 tuples, the running times of *MicroRandLS* and *MicroLAHLS-4* were around 1100 and 1500 seconds respectively.

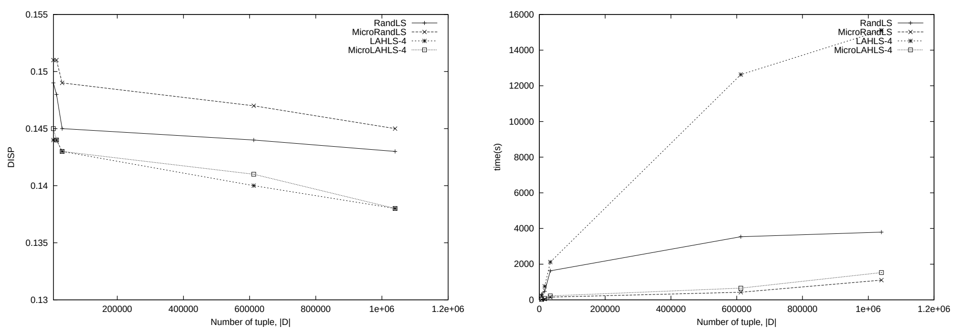
To see the effect that an existential constraint has on the clustering, we look at the output of *MicroLAHLS-4* in Figure 4 which shows a synthetic dataset with $M = 8$. The clustering was done with the existential constraint of "$count(Cl_i) \geq 812$" imposed. Since the actual clusters that were generated near the top-left corner generally contained less than 812 points, there was a shift of cluster centers from the top-left corner towards the dense region at the bottom-right corner.

To summarize, our experimental results show that the our algorithm is effective for constrained clustering. Among the heuristics, micro-cluster sharing clearly delivers good efficiency and scalability. The gain in efficiency far offsets the small loss in quality. Finally, the look-ahead heuristic with small l (e.g., 4) appears to be the best candidate for pivot movement.



M	k	$\|D\|$	c	No. of Micro-clusters
4	16	6800	212	1410
5	25	16250	325	2523
6	36	33300	462	4050
7	49	61250	625	7079
8	64	104000	812	8253

Fig. 4. 64 Clusters with ≥ 812 Objects Each. **Fig. 5.** Parameters for Synthetic Dataset.



(a) Average DISP per object against $|D|$ (b) Running time against $|D|$

Fig. 6. Performance of Various Algorithms as $|D|$ Varies.

8 Discussion: Handling Other SQL Aggregate Constraints

We have presented a cluster refinement algorithm which handles a *single existential constraint*. In this section, we first examine how the algorithm can be extended to handle constraints containing single SQL aggregate, where the constrains are classified into five classes (see Table 1) based on their behavior with respect to constrained clustering: *existential, existential-like, universal, averaging*, and *summation*.

1. Universal constraints: These are constraints in which a specific condition must be satisfied by *every* object in a cluster. For example, $min(\{O_i[A_j]|O_i \in Cl_i\}) \geq c$ requires that every object's A_j-value be $\geq c$. This can be reduced to the UC problem as discussed for constraints on individual objects in Section 2.

2. Existential-like constraints: These constraints are similar in nature to existential constraints, and our algorithm can handle them with simple modification. For

	$<$ or $\leq$	$\neq$	$=$	$>$ or $\geq$
min	existential	existential-like	existential-like	universal
max	universal	existential-like	existential-like	existential
count	existential-like	existential-like	existential	existential
avg	averaging	averaging	averaging	averaging
sum	summation	summation	summation	summation

Table 1. A Classification of SQL Constraints

example, $count(Cl_i) \leq c$ is an existential-like constraint. Instead of moving surplus objects around, the objective here is to move "holes" around to achieve the maximum reduction in $DISP$. If a cluster Cl_i contains m objects, $m < c$, it has $c - m$ holes, meaning that $c - m$ objects can still be moved into it. When an object is moved from Cl_j into Cl_i, a hole is moved from Cl_i into Cl_j. Correspondingly, a *hole movement graph* can be generated which could be used to guide movement in the locality graph.

3. Averaging and Summation constraints. For these kinds of constraints, even computing an initial solution is an NP-hard problem similar to a bin-packing or knapsack problem [GJ79]. Handling general averaging and summation constraints in clustering is an interesting problem for future work.

Finally, we consider the situation when there are multiple conjunctive existential constraints. The local search algorithm can easily be modified to handle existential constraints when the sets of pivot objects for these constraints do not overlap. The algorithm can then set up a different pivot movement graph for each constraint, and move the pivot objects in different graphs independently. However, for situations where the sets of pivot objects do overlap, again we can show even computing an initial solution is NP-hard. Handling multiple general existential constraints is another interesting problem for future work.

9 Conclusions

In this paper, we introduced and studied the constrained clustering problem, a problem which arises naturally in practice, but barely addressed before. A (constrained) cluster refinment algorithm is developed, which includes two phases of movement in a clustering locality graph: *pivot movement* and *deadlock resolution*. Our experimental results show that both phases are valuable. To scale up the algorithm for large databases, we proposed a micro-cluster sharing strategy whose effectiveness is also verified by our experiments. Our algorithm can also be extended to handle some other kinds of constraints, however, handling general averaging and summation constraints, as well as handling general multiple existential constraints, are interesting topics for future research. Thanks to a reviewer of this paper, we have come to know of a recent study by Bradley et al. [BBD00] on a version of constrained clustering problem similar to ours. Unlike us, their main motivation is using cardinality constraints for better quality clustering. Scalability and applicability to other types of constraints are not addressed. Despite these differences, a quantitative comparison between the two approaches would be an interesting future work.

References

[AGGR98] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *SIGMOD'98.*

[BBD00] P. Bradley, K. P. Bennet, and A. Demiriz. Constrained k-means clustering. In *MSR-TR-2000-65*, Microsoft Research, May 2000.

[BFR98] P. Bradley, U. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *KDD'98.*

[EKSX96] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases. In *KDD'96.*

[GJ79] M. Garey and D. Johnson. *Computers and Intractability: a Guide to The Theory of NP-Completeness.* Freeman and Company, New York, 1979.

[HaKa00] J. Han and M. Kamber *Data Mining: Concepts and Techniques.* Morgan Kaufmann, 2000.

[KHK99] G. Karypis, E.-H. Han, and V. Kumar. CHAMELEON: A hierarchical clustering algorithm using dynamic modeling. *COMPUTER*, 32:68–75, 1999.

[KMR97] D. Karger, R. Motwani, and G. D. S. Ramkumar. On approximating the longest path in a graph. *Algorithmica*, 18:99–110, 1997.

[KPR98] J. M. Kleinberg, C. Papadimitriou, and P. Raghavan. A microeconomic view of data mining. *Data Mining and Knowledge Discovery*, 2:311–324, 1998.

[KR90] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: an Introduction to Cluster Analysis.* John Wiley & Sons, 1990.

[LNHP99] L. V. S. Lakshmanan, R. Ng, J. Han, and A. Pang. Optimization of constrained frequent set queries with 2-variable constraints. In *SIGMOD'99.*

[NH94] R. Ng and J. Han. Efficient and effective clustering method for spatial data mining. In *VLDB'94.*

[NLHP98] R. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *SIGMOD'98.*

[STA97] D. B. Shmoys, E. Tardos, and K. Aardal. Approximation algorithms for facility location problems. In *STOC'97.*

[TNLH00] A. K. H. Tung, R. Ng, L. Lakshmanan, and J. Han. Constraint-based clustering in large databases. www.cs.sfu.ca/pub/cs/techreports/2000/CMPT2000-05.pdf.

[WYM97] W. Wang, J. Yang, and R. Muntz. STING: A statistical information grid approach to spatial data mining. In *VLDB'97.*

[ZRL96] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: an efficient data clustering method for very large databases. In *SIGMOD'96.*