Efficient Incremental Constrained Clustering

Ian Davidson
SUNY - Albany
Computer Science
Albany, NY 12222
davidson@cs.albany.edu

Martin Ester
Simon Fraser University
Computing Science
Burnaby, BC V5A 1S6
ester@cs.sfu.ca

S. S. Ravi
SUNY - Albany
Computer Science
Albany, NY 12222
ravi@cs.albany.edu

ABSTRACT

Clustering with constraints is an emerging area of data mining research. However, most work assumes that the constraints are given as one large batch. In this paper we explore the situation where the constraints are incrementally given. In this way the user after seeing a clustering can provide positive and negative feedback via constraints to critique a clustering solution. We consider the problem of *efficiently* updating a clustering to satisfy the new and old constraints rather than re-clustering the entire data set. We show that the problem of incremental clustering under constraints is NP-hard in general, but identify several sufficient conditions which lead to efficiently solvable versions. These translate into a set of rules on the types of constraints that can be added and constraint set properties that must be maintained. We demonstrate that this approach is more efficient than re-clustering the entire data set and has several other advantages.

Categories and Subject Descriptors

H.2.8 [**Information Systems**]: Database management. Data mining.

General Terms

Algorithms, Experimentation, Theory

Keywords

Clustering, Constraints, Algorithms

1. INTRODUCTION AND MOTIVATION

The last five years have seen extensive work on incorporating instance-level constraints into clustering methods [14, 1, 2]. Constraints provide guidance about the desired partition and make it possible for clustering algorithms to perform better, sometimes dramatically. Instance-level constraints specify that two items must be placed in the same cluster (must-link, ML) or in different clusters (cannot-link, CL). This semi-supervised approach has led to improved performance for several UCI data sets as well as for real-world applications, such as person identification from surveillance camera clips [2], noun phrase coreference resolution, GPS-based map refinement [14] and landscape detection from hyperspectral data [11].

However, most constrained clustering work involves batch style specification of the constraints and the subsequent running of the clustering algorithm. In many applications, it may be reasonable for constraints to be incrementally added or removed based on feedback from a user. This can be viewed as the user *critiquing* a clustering and can be repeated often until the user is happy with the resulting clustering. Furthermore, it is known that this approach of selectively choosing constraints produces significantly better results than randomly choosing constraints. Consider the situation of document clustering as described by Cohn, Caruana and McCallum [3]. After the clustering is performed, the user has the option of critiquing the clustering by implicit feedback ("a document does not belong in this cluster") and explicit feedback ("a document belongs here" or "these documents should be together/apart"). This incorporation of a user into the clustering loop is analogous to active learning, except that the constraints are chosen by a user rather than by a machine. Cohn et al. [3] explore the use of this type of active constraint generation in clustering. In the area of document clustering using the Reuter's newsgroup data, they simulate actively chosen constraints (by picking two instances placed in the same cluster but with known different categories/labels). Ten constraints chosen using this approach produces as good results as using between 300% to 600% more constraints that are randomly chosen. Furthermore, Davidson, Wagstaff and Basu [8] have recently shown that there is a significant variation in the benefits (when measured by purity on the class labels) provided by different sets of constraints. This was shown to be the case despite each constraint set having the same number of constraints from the same source (set of labels). This provides further reason to believe that user-specified constraints are better than randomly chosen constraints.

The works of Cohn et al. [3] and Davidson et al. [8] illustrate that incremental constrained clustering offers tremendous potential improvement over batch style constraint processing and that not all constraint sets are equal in terms of their effect on the final solution. However, the work of Cohn et al. required the clustering algorithm to be *re-run after each set of constraints is added*, an approach we refer to as

KDD'07, August 12–15, 2007, San Jose, California, USA.

non-incremental constrained clustering or *total reclustering*. This is an extremely time consuming exercise if the data set is large. In addition, the user may have liked the existing clustering and there is no guarantee that a similar clustering would be found after reapplying the algorithm. We provide experimental evidence to support of this claim.

Contributions of this paper. The purpose of this work is to investigate the conditions under which incremental clustering with constraints can be carried out efficiently with large data sets. To do this, we allow the addition and removal of constraints and try to find clusterings that can be obtained by making a small number of changes to a given clustering. We begin in section 2 by defining several versions of the incremental constrained clustering problem. These problems encompass reasonable methods of specifying feedback in the form of constraints. In general, incrementally adding constraints is intractable; further, incrementally removing constraints is intractable when a reasonable secondary objective such as minimizing the partition diameter is specified. These results are in shown in section 3. Because of these general intractability results, our focus is on finding sufficient conditions which give rise to restricted versions of the problem that can be solved efficiently. These sufficient conditions can be used to provide instructions to a user regarding the properties to be satisfied by constraints so that a given partition can be efficiently updated to satisfy the new constraints. Our sufficient conditions are described in Sections 4.1 and 4.2, and a greedy update algorithm that uses these conditions is described in Section 5. Section 6 presents experimental results to show that our incremental constrained algorithm is more efficient than re-running the batch style constrained clustering algorithm and even produces better quality results compared to batch style constrained clustering algorithms that attempt to satisfy all constraints.

2. PROBLEM DEFINITIONS

Assumptions. To provide a proper perspective for our contributions, we mention the assumptions used in this work. Firstly, our earlier work on the feasibility problem [4, 5, 7] addressed the question of determining whether there is a partition that satisfies a given set of constraints. In this paper, we assume that a clustering satisfying all the initial constraints (C) is given and that the addition of new constraints (C') may give rise to a combined set of constraints for which no feasible clustering exists for the given value of k (the number of clusters). We also assume that the value of k cannot be changed when constraints are added or removed. Without loss of generality, we study the addition of only one constraint at a time. The addition of multiple constraints can be carried out by repeating the procedure for each constraint. Finally, we assume that adding a new constraint does not cause an obvious contradiction such as $ML(x, y)$ and $CL(x, y)$.

In an ideal situation, the user should be able to specify several different forms of feedback such as:

(a) An instance x does not belong in cluster Q.

(b) An instance x should be in cluster Q.

(c) Two instances x and y must be in the same cluster.

(d) Two instances x and y shouldn't be in the same cluster.

Feedback types (a) and (b) provide label information for an instance while (c) and (d) provide constraints on two instances. It should be noted that it is possible to unambiguously translate a set of labels on instances to a set of constraints but the converse is not true. For this reason, we assume that all feedback from the user is in the form of must-link and cannot-link constraints. While these constraints will not be able to efficiently encode all types of feedback (e.g. limits on cluster sizes), they can model many useful forms of feedback.

We now formally define the incremental clustering problems addressed in this paper. The incremental nature of the problems are with respect to a) *adding* constraints and b) *removing* constraints. When a change (i.e., the addition or deletion of constraints) is specified, the goal is to efficiently update a given clustering, **without running the base clustering algorithm**. In updating a partition, one may also be interested in optimizing a suitable objective function (e.g. the objective function used by the algorithm that produced the original clustering) or the number of changes to the given partition.

Throughout this paper, a clustering is a partition of the given set of points. We use the term "k-clustering" to mean a partition with k subsets. Several objective functions are known (e.g. vector quantization error, partition diameter, partition purity) for measuring the quality of a partition. Given a partition Π, we use $f(\Pi)$ to denote the quality of the partition as given by the objective function. Some objective functions represent minimization objectives (e.g. vector quantization error, partition diameter) while others (e.g. partition purity) are maximization objectives. We will formulate the incremental clustering problems assuming that f is a minimization objective. We will also indicate the changes to the formulation when f is a maximization objective.

We begin by formulating the incremental clustering problem under the *addition* of new constraints. These formulations assume that the new clustering (when it exists) should also improve the objective function. We will comment on this issue after the problem specification.

PROBLEM 2.1. ***Incremental Clustering under Constraint Addition.*** *Let S be a set of points and let C be a set of constraints. Let Π be a k-clustering of S satisfying all the constraints in C. Let C' be a new set of constraints such that $C \cap C' = \emptyset$. Is there a k-clustering Π' of S such that (i) Π' satisfies all the constraints in $C \cup C'$ and (ii) $f(\Pi') \leq f(\Pi)$? If so, find a partition Π^* for which $f(\Pi^*)$ is a minimum among all partitions that satisfy conditions (i) and (ii).*

When constraints are added, the most desirable outcome is that there is a feasible partition for the new set of constraints and the partition improves the value of the objective function. However, this may not be the case in general. When there is no feasible partition for the constraint set $C \cup C'$, a system can ask the user to choose a different set of constraints. Another possibility is that there are feasible partitions for the new constraint set, but none of these partitions improves the objective function value. In such a case, an algorithm would try to produce a partition Π' such that the increase in the value of the objective function is as small as possible. It is again up to the user to decide whether or not the new constraints should be used.

In the statement of Problem 2.1 if f is a maximization objective, the condition "$f(\Pi') \leq f(\Pi)$" would be replaced by "$f(\Pi') \geq f(\Pi)$" and one would seek an optimal partition Π^* for which $f(\Pi^*)$ is a maximum among all partitions that satisfy the conditions.

We now define the natural analog of the constraint addition problem, namely the constraint removal problem.

PROBLEM 2.2. ***Incremental Clustering under Constraint Removal.*** *Let S be a set of points and let C be a set of constraints. Let Π be a k-clustering of S satisfying all the constraints in C. Let $C' \subset C$ be a subset constraints to be removed. Find a k-clustering Π^* of S such that $f(\Pi^*)$ is a minimum among all the k-clusterings that satisfy the constraints in $C - C'$.*

The formulation of the problem under constraint removal is slightly different from that of constraint addition since feasibility is not an issue in the former case; the original partition will itself satisfy the new set of constraints. So, the focus is on optimizing the objective function value.

3. WORST CASE COMPLEXITY RESULTS

This section presents the following results.

(a) The problem of determining whether there is a feasible solution under the addition of a single ML or CL constraint is computationally intractable. These results do not depend on optimizing the objective function. They are discussed in Sections 3.1 and 3.2.

(b) As mentioned earlier, the feasibility problem is trivial under constraint removal. However, we show that if the goal is to delete a constraint and minimize the partition diameter, the problem is computationally intractable. This result is presented in Section 3.3.

We have also obtained complexity results for other forms of incremental clustering such as removing *points* (not constraints) to reduce cluster diameter. These results are not shown here due to space reasons. We believe that these are somewhat specialized versions of the incremental clustering problem and that practitioners are likely to be more interested in addition and removal of constraints.

The remainder of this section, which proves the results mentioned in (a) and (b) above, can be skipped on first reading of this paper without loss of flow.

3.1 Incremental Clustering Under the Addition of a Must-Link Constraint

Given a set of points, a set of constraints and a partition of the point set into k subsets such that all the given constraints are satisfied, we want to determine whether the addition of a new ML constraint can cause infeasibility. In this section, we examine the complexity of this problem. We begin with a precise formulation of the problem.

Incremental Feasibility Testing: Adding an ML Constraint (IFT-ML)

Instance: A set $S = \{s_1, s_2, \ldots, s_n\}$ of n points; an integer k, $1 \leq K \leq n$; a set C of constraints; a partition Π of S into k subsets such that Π satisfies all the constraints in C; a constraint $\mathrm{ML}(s_i, s_j)$ which is not in C.

1. Let $G(V, E)$ be the given graph, with $V = \{v_1, v_2, \ldots, v_n\}$.
2. Construct graph $G_1(V_1, E_1)$ as follows.

   (a) Let $V_A = \{a_1, a_2, \ldots, a_n\}$ and $V_B = \{b_1, b_2, \ldots, b_n\}$. Let $V_1 = V \cup V_A \cup V_B$.

   (b) Let E' denote the set of edges $\{\{a_i, v_i\}, \{b_i, v_i\} : 1 \leq i \leq n\}$. Let E'' denote the set of edges $\{\{a_i, b_i\} : 1 \leq i \leq n\}$. Let $E_1 = E \cup E' \cup E''$.

Figure 1: A Graph Construction Procedure

Required: Determine whether there is a partition of S into k subsets such that all the constraints in $C \cup \{\mathrm{ML}(s_i, s_j)\}$ are satisfied. If so, output one such partition Π'.

The main result of this section is that unless $\mathbf{P} = \mathbf{NP}$, there is no efficient algorithm for the IFT-ML problem. Before we can prove this result, we need to introduce some preliminary definitions and results.

3.1.1 Preliminary Definitions and Results

Recall that the graph 3-colorability problem is **NP**-complete [9]. In fact, the problem remains **NP**-complete even when restricted to graphs in which the maximum node degree is 4 [9]. We refer to this restricted version of the graph 3-colorability problem as the **Restricted 3-Colorability Problem** (R3CP). The following result is proven in [9].

THEOREM 3.1. *The Restricted 3-Colorability Problem is **NP**-complete.* ■

We also need another result from graph theory. This well known result, called **Brooks's Theorem**, gives an upper bound on the number of colors needed for a graph in terms of the maximum node degree. The proof of the following theorem can be found in [13].

THEOREM 3.2. ***[Brooks's Theorem]*** *Any undirected graph with a maximum node degree of Δ can be colored using at most $\Delta+1$ colors. Moreover, such a coloring can be obtained in polynomial time.* ■

Let $G(V, E)$ be an undirected graph with $V = \{v_1, v_2, \ldots, v_n\}$. Figure 1 describes a procedure for constructing another graph G_1 from G. Informally, this construction adds two new nodes for each node of G and adds three edges that connect the three nodes into a complete graph. An example of this construction is shown in Figure 2. Some key properties of this construction are shown in the following lemma.

LEMMA 3.1. *Suppose G is an undirected graph with n nodes and a maximum node degree of 4. Let G_1 be the graph obtained from G using the construction described in Figure 1. The following properties hold.*

(a) G_1 is 5-colorable. Moreover, a 5-coloring of G_1 can be constructed in polynomial time.

(b) There is a 5-coloring of G_1 in which all the nodes in $V_A = \{a_1, a_2, \ldots, a_n\}$ have the same color and all the

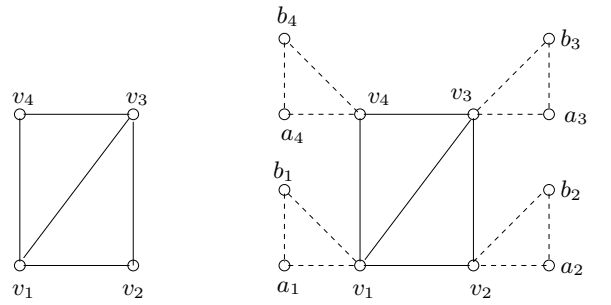


Figure 2: An example for the graph construction procedure described in Figure 1. (The given graph G is shown in (i) and the graph G_1 resulting from the construction is shown in (ii). The new edges added to produce G_1 are shown as dashed lines.)

nodes in $V_B = \{b_1, b_2, \ldots, b_n\}$ have the same color if and only if G is 3-colorable.

Proof:
Part (a): Note that G has a maximum node degree of 4. Thus, by Brooks's theorem, G is 5-colorable and one such 5-coloring can be obtained in polynomial time. A 5-coloring of G_1 can be constructed efficiently from that of G as follows. Consider any node v_i of G and let j denote the color assigned to v_i. Choose two different colors other than j and assign them to a_i and b_i respectively. By repeating this process for each vertex of G, we obtain a 5-coloring of G_1.
Part (b):
If part: Suppose G is 3-colorable, and let the colors used be $\{1, 2, 3\}$. Use color 4 for all the nodes in V_A and color 5 for all the nodes in V_B. Clearly, this is a 5-coloring of G_1 in which all the nodes in V_A have the same color and all the nodes in V_B have the same color.
Only If part: Suppose there is a 5-coloring of G_1, using colors $\{1, 2, 3, 4, 5\}$, in which all the nodes in V_A have the same color and all the nodes in V_B have the same color. Note that the color assigned to the nodes in V_A must be different from the one assigned to the nodes in V_B since G_1 has the edge $\{a_1, b_1\}$. So, let 4 and 5 denote the colors assigned to the nodes in V_A and V_B respectively. Now, consider any node v_i of G. Since v_i is adjacent to both a_i and b_i, colors 4 and 5 cannot be used for v_i. In other words, each node of G must be colored 1, 2 or 3; that is, G is 3-colorable. ■

3.1.2 *Complexity of the* IFT-ML *Problem*

THEOREM 3.3. *Unless **P** = **NP**, there is no polynomial time algorithm for* IFT-ML *problem.*

Proof: Suppose there is a polynomial time algorithm $\mathcal{A}$ for the IFT-ML problem. We will show that $\mathcal{A}$ can be used to solve the Restricted 3-coloring Problem (R3CP) in polynomial time. Since R3CP is **NP**-complete (Theorem 3.1), this would contradict the assumption that $\mathbf{P} \neq \mathbf{NP}$.

Consider any instance of the R3CP problem with $G(V, E)$ being a graph in which the maximum node degree is 4.

1. Let $G(V, E)$ be the given graph with maximum node degree 4, with $V = \{v_1, v_2, \ldots, v_n\}$.
2. Construct graph $G_1(V_1, E_1)$ from G as described in Figure 1. Let $V_A = \{a_1, a_2, \ldots, a_n\}$, $V_B = \{b_1, b_2, \ldots, b_n\}$ and $E_1 = \{e_1, e_2, \ldots, e_m\}$.
3. Create a set of points $S = X \cup Y \cup Z$, where $X = \{x_1, x_2, \ldots, x_n\}$, $Y = \{y_1, y_2, \ldots, y_n\}$ and $Z = \{z_1, z_2, \ldots, z_n\}$. (The sets X, Y and Z are in one-to-one correspondence with sets V, V_A and V_B respectively. Coordinates are not specified for the points in S since they play no role in the algorithm.)
4. For each edge $e_j \in E_1$, create the constraint $c_j = \mathrm{CL}(x, y)$, where x and y are the points corresponding to the two nodes joined by e_j, $1 \leq j \leq m$. Initialize the constraint set C to $\{c_1, c_2, \ldots, c_m\}$.
5. Find a 5-coloring of G_1 as explained in the proof of Lemma 3.1. Let W_r be the set of nodes of G_1 such that all nodes in W_r have color r, $1 \leq r \leq 5$. Create a partition Π of S into 5 subsets S_1, S_2, S_3, S_4 and S_5, where S_r has the points corresponding to the nodes in W_r, $1 \leq r \leq 5$. (Note that Π satisfies all the constraints in C.)
6. Let L denote the list of the following $2n-2$ ML constraints: $\langle \mathrm{ML}(y_1, y_2), \mathrm{ML}(y_2, y_3), \ldots, \mathrm{ML}(y_{n-1}, y_n), \mathrm{ML}(z_1, z_2), \mathrm{ML}(z_2, z_3), \ldots, \mathrm{ML}(z_{n-1}, z_n)\rangle$. Let L_i denote the i^{th} constraint in L, $1 \leq i \leq 2n-2$.
7. **for** $i = 1$ **to** $2n-2$ **do**
   (a) Execute Algorithm $\mathcal{A}$ on S, with constraint set C and partition Π to determine whether there is a feasible solution when constraint L_i is added.
   (c) **if** Algorithm $\mathcal{A}$ returns "Yes" along with new partition Π'
   then
   Let $C = C \cup \{L_i\}$ and $\Pi = \Pi'$.
   else
   Print "G is not 3-colorable" and **stop**.
8. Print "G is 3-colorable".

Figure 3: Algorithm $\mathcal{B}$ Used to Prove Theorem 3.3

We refer to the algorithm in Figure 3 as $\mathcal{B}$. This algorithm constructs the graph G_1 from G (as described in Figure 1), creates a set S of points corresponding to the nodes of G_1, and uses the edges of G_1 to create the initial set C consisting of CL constraints. The algorithm produces an initial partition of S into 5 subsets using the result of Lemma 3.1(a). The algorithm then successively adds the ML constraints $\text{ML}(y_1, y_2)$, $\text{ML}(y_2, y_3)$, ... $\text{ML}(y_{n-1}, y_{n-1})$, $\text{ML}(z_1, z_2)$, $\text{ML}(z_2, z_3)$, ... $\text{ML}(z_{n-1}, z_{n-1})$, one at a time, and invokes Algorithm $\mathcal{A}$ to determine whether there is a feasible solution after each addition. In terms of G_1, it can be seen that the effect of above collection of ML constraints is to force all nodes in V_A to have the same color and all nodes in V_B to have the same color. Thus, from Lemma 3.1(b), it follows that Algorithm $\mathcal{A}$ can produce a feasible solution satisfying all the original constraints and the new ML constraints if and only if G is 3-colorable.

Thus, Algorithm $\mathcal{B}$ correctly decides whether G is 3-colorable. In Figure 3, it is easy to see that all steps except Step 7 can be carried out in polynomial time. In Step 7, Algorithm $\mathcal{B}$ makes at most $2n - 2$ calls to Algorithm $\mathcal{A}$. Thus, if Algorithm $\mathcal{A}$ runs in polynomial time, then so does Algorithm $\mathcal{B}$. In other words, we have a polynomial time algorithm for the R3CP problem. This contradicts our assumption that $\mathbf{P} \neq \mathbf{NP}$ and completes the proof of Theorem 3.3. ■

3.2 Incremental Clustering Under the Addition of a Cannot-Link Constraint

In this section, we present our complexity result for incremental clustering when a CL constraint is added. The problem formulation is as follows.

Incremental Feasibility Testing: Adding a CL Constraint (IFT-CL)

Instance: A set $S = \{s_1, s_2, \ldots, s_n\}$ of n points; an integer k, $1 \leq K \leq n$; a set C of constraints; a partition Π of S into k subsets such that Π satisfies all the constraints in C; a constraint $\text{CL}(s_i, s_j)$ which is not in C.

Required: Determine whether there is a partition of S into k subsets which satisfies all the constraints in $C \cup \{\text{CL}(s_i, s_j)\}$. If so, output one such partition Π'.

The following theorem points out the difficulty of obtaining an efficient algorithm for the IFT-CL problem.

THEOREM 3.4. *Unless* $\boldsymbol{P = NP}$*, there is no polynomial time algorithm for the* IFT-CL *problem.*

Proof: Suppose there is a polynomial time algorithm $\mathcal{A}_1$ for the IFT-CL problem. We will show that $\mathcal{A}_1$ can be used to devise a polynomial time algorithm for the k-coloring problem. This would contradict the assumption that $\mathbf{P} \neq \mathbf{NP}$.

Let $G(V, E)$ and integer k represent the given instance of the k-coloring problem. We refer to the algorithm shown in Figure 4 as $\mathcal{B}_1$. Algorithm $\mathcal{B}_1$ starts with an empty set C of constraints and an arbitrary partition of the set S into k subsets. (Since the constraint set C is empty, any partition of S is a feasible solution.) For each edge of the graph, the algorithm creates a new CL constraint and invokes Algorithm $\mathcal{A}_1$ to determine whether there is a feasible solution when the new constraint is added. We now show that $\mathcal{B}_1$ correctly decides whether G is k-colorable.

Suppose Algorithm $\mathcal{B}_1$ outputs the message "G is k-colorable". Then, from the description in Figure 4, Algorithm $\mathcal{A}_1$ produced a feasible partition into k subsets after all the CL constraints corresponding to the edges of G were added. Let S_1, S_2, ..., S_K denote the resulting subsets. Consider the coloring of G obtained by assigning color j to all the nodes corresponding to the points in S_j, $1 \leq j \leq K$. This coloring uses k colors. Further, for any edge $\{v_x, v_y\}$ of G, the constraint $\text{CL}(s_x, s_y)$ ensures that s_x and s_y are in different subsets of the partition. In other words, nodes v_x and v_y have different colors. Thus, we have a valid k-coloring of G.

Suppose Algorithm $\mathcal{B}_1$ outputs the message "G is not k-colorable". Thus, at some stage, Algorithm $\mathcal{A}_1$ must have returned "No" in Step 3(c) of Figure 4. We prove by contradiction that there is no valid k-coloring of G. Suppose G is k-colorable, and let 1, 2, ..., k denote the colors used. Thus, there is a partition of the node set V into k subsets V_1, V_2, ..., V_K such that V_j is the set of all nodes assigned color j, $1 \leq j \leq K$. Consider the partition of S into k subsets S_1, S_2, ..., S_K, where S_j contains all the points corresponding to the nodes in V_j, $1 \leq j \leq K$. Clearly, this partition satisfies all the CL constraints added by Algorithm $\mathcal{B}_1$. Thus, Algorithm $\mathcal{A}_1$ cannot return "No" at any stage. This contradiction shows that G has no valid k-coloring.

Thus, Algorithm $\mathcal{B}_1$ correctly decides whether G is k-colorable. In Figure 4, it is easy to see that all steps except Step 3 can be carried out in polynomial time. In Step 3, Algorithm $\mathcal{B}_1$ makes at most $m = |E|$ calls to Algorithm $\mathcal{A}_1$. Thus, if Algorithm $\mathcal{A}_1$ runs in polynomial time, then so does Algorithm $\mathcal{B}_1$. In other words, we have a polynomial time algorithm for the k-coloring problem. This contradicts our assumption that $\mathbf{P} \neq \mathbf{NP}$ and completes the proof of Theorem 3.4. ■

3.3 Removing Constraints to Reduce Partition Diameter

We now discuss the flip-side problem of removing constraints. Since we are given a partition that already satisfies a given set of constraints, such a partition will trivially also satisfy any subset of these constraints. Instead we shall explore whether there exists a new partition which minimizes the clustering diameter. We begin by first defining the clustering (partition) diameter.

DEFINITION 3.1. *Suppose* $S = \{s_1, s_2, \ldots, s_n\}$ *is a set of* n *points with a distance* $d(s_i, s_j)$ *for each pair of pointe* s_i *and* s_j *in* S*. Let* Π *be a partition of* S *into* k *subsets (clusters)* S_1, S_2, ..., S_K*. The* ***diameter of cluster*** S_i*, denoted by* $\text{DIA}(S_i)$*,* $1 \leq i \leq K$*, is given by*

$$\text{DIA}(S_i) \;=\; \max\{d(s_x, s_y) \;:\; s_x, s_y \in S_i\}.$$

The ***diameter of the partition*** Π*, denoted by* $\text{DIA}(\Pi)$*, is given by*

$$\text{DIA}(\Pi) \;=\; \max\{\text{DIA}(S_i) \;:\; 1 \leq i \leq K\}.$$

Pruning Constraints to Decrease Diameter (PCDD)

Instance: A set $S = \{s_1, s_2, \ldots, s_n\}$ of n points; distance $d(s_i, s_j)$ for each pair of pointe s_i and s_j in S; an integer k, $1 \leq K \leq n$; a set C of constraints; a partition Π of S into k subsets such that Π satisfies all the constraints in C and $\text{DIA}(\Pi) = \sigma_1$; a subset $C' \subseteq C$ of constraints and a number $\sigma_2 < \sigma_1$.

1. Let $G(V, E)$ be the given graph, with $V = \{v_1, v_2, \ldots, v_n\}$ and $E = \{e_1, e_2, \ldots, e_m\}$.
2. Create an arbitrary set $S = \{s_1, s_2, \ldots, s_n\}$ of n points. (Note that S is one-to-one correspondence with the node set V. Coordinates for the points in S are not specified since they play no role in the algorithm.) Initially, the constraint set C is empty. Create an arbitrary partition of S into k subsets. Let Π denote this partition.
3. **for** $i = 1$ **to** m **do**
   (a) Let edge e_i join nodes v_x and v_y. Create a new constraint $c = \text{CL}(s_x, s_y)$.
   (b) Execute Algorithm $\mathcal{A}_1$ on S, with constraint set C, partition Π and new ML constraint c to determine whether there is a feasible solution.
   (c) **if** Algorithm $\mathcal{A}_1$ returns "Yes" along with new partition Π'
   then
   Let $C = C \cup \{c\}$ and $\Pi = \Pi'$.
   else
   Print "G is not k-colorable" and **stop**.
4. Print "G is k-colorable".

Figure 4: Algorithm Used to Prove Theorem 3.4

Question: Is there a partition Π' of S into k blocks such that Π' satisfies all the constraints in $C - C'$ and $\text{DIA}(\Pi') \leq \sigma_2$?

THEOREM 3.5. *The* PCDD *problem is* ***NP****-complete. Moreover, the result holds even when the given constraint set contains only ML constraints and the number of constraints to be removed is 1.*

Proof Idea: The proof involves a reduction from k-coloring and is omitted due to space reasons.

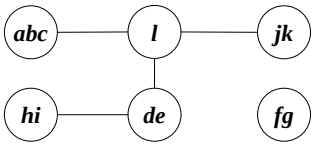
4. EASY INSTANCES OF INCREMENTAL CONSTRAINED CLUSTERING PROBLEMS

In this section we introduce several sufficient conditions that despite our previous worst case results, give rise to situations where there are efficient algorithms to find a feasible clustering to satisfy the new and old set of constraints. We begin by first describing the basic conditions and then move on to non-trivial conditions. In section 5 we put these results together into a greedy algorithm that we experimentally verify in section 6 and compare its performance against non-incremental constrained clustering.

4.1 Basic Sufficient conditions

We present the following basic situations with little explanations and no proofs. Without loss of generality we discuss a single constraint on points x and y. Recall that a clustering Π that satisfies the constraint set C is given and that the additional constraint is denoted by C'.

Figure 5: A *constraint graph* for *ML(a,b), ML(a,c), ML(d,e), ML(f,g), ML(h,i), ML(j,k), CL(a,l), CL(l,j), CL(d,i), CL(d,l)*



- The constraint in C' is on a pair of points that are not already involved in a constraint in C. In this situation, if the constraint is not already satisfied, we move the constrained points (x and y) together (for ML) or apart (for CL) in such a way to minimize the objective function f. For ML and CL constraints, this would use respectively $O(k)$ and $O(k^2)$ evaluations of the function f.
- The constraint in C' involves a pair of points such that only one of the points is involved in an existing constraint in C. Since one of the points is unconstrained, it can be easily moved to satisfy the constraint. For an ML constraint, this does not need any evaluation of the function f. For a CL constraint, this uses $O(k)$ evaluations of f.
- All of the constraints in C and C' are must-link constraints. In such a situation we can efficiently recompute the transitive closure.

4.2 Non-Trivial Sufficient Conditions

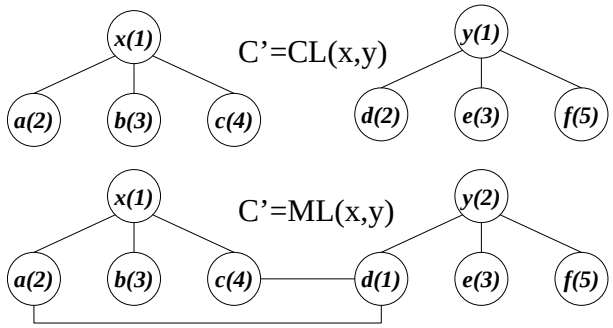
The non-trivial situations occur when both constrained points x and y are already involved in constraints in C. This is precisely the situation that is exploited in proving our worst case results. For these non-trivial situations we make use of several sufficient conditions which make the incremental clustering problem easy. We now describe these sufficient conditions and the implications on the limitations of what constraints can be included in C'.

4.2.1 Brooks's Theorem

Consider clustering under the set of constraints: $ML(a, b)$, $ML(a, c)$, $ML(d, e)$, $ML(f, g)$, $ML(h, i)$, $ML(j, k)$, $CL(a, l)$, $CL(l, j)$, $CL(d, i)$ and $CL(d, l)$ as graphically shown in Figure 5. The edges represent CL constraints and must-linked points are represented by a single node. We refer to such a graph as a *constraint graph*. Suppose we wish to cluster the data for $k = 3$. Proceeding in the order of the points in the data set (alphabetically), there could be an assignment of abc to cluster 1, de to cluster 2, jk, hi and fg to cluster 3, but then point l can't be assigned to any cluster feasibly. It is not that the graph is not 3-colorable, rather the ordering of the nodes presents a problem. This is of pragmatic importance [7] since the ordering of the points as processed by clustering algorithms is typically fixed apriori and does not change.

Brooks's theorem (see Theorem 3.2) points out that if the maximum node degree of a graph G is Δ and $k \geq \Delta + 1$, then a k-coloring of the graph is easy to obtain using *any* linear ordering of the nodes. This means that if the maximum number of CL-constraints involving the same point is at most $k - 1$, then the feasibility problem is guaranteed to be easy; otherwise, the feasibility problem *may be* difficult.

Figure 6: A graphical representation of the situation where the constraint to add is either a ML or CL constraint for $k = 5$. The number in parentheses is the cluster number the point is assigned to. We discuss single constraints without loss of generality.



The above discussion points out one way of making the incremental constrained clustering problem easy. We can add constraints without restriction except to make sure that no single point is involved in k or more CL constraints, including given and entailed constraints.

We can use this sufficient condition to make the incremental addition of a CL constraint easy, but the addition of an ML constraint may still involve a large number of cluster reassignments. Why this is the case can be explained by examining figure 6. Consider the top graph. Both x and y are already constrained and are in the same cluster. The new constraint to add is $\text{CL}(x, y)$. Since we want to use Brooks's theorem, both x and y will have at least one "free" color/cluster-id. In our example, we can assign x to cluster 5 or y to cluster 4, whichever choice optimizes the chosen objective function f. In general, this operation may use $O(k^2)$ evaluations of the function f. However, if the new constraint to add is $\text{ML}(x, y)$, no such easy reassignment may exist, as can be seen by considering the lower diagram in figure 6. Clearly, x cannot be placed in y's cluster and vice-versa. The only solution is to place x and y together in cluster 3, 4 or 5 and then assign those points in the chosen cluster to which x and y are cannot-linked to another cluster, while making sure that none of *their* constraints are violated.

Note that Brooks's theorem states that a $\Delta + 1$ coloring is possible regardless of the order in which nodes are considered. However, there may exist a coloring using far less the $\Delta + 1$ colors. The notion of *inductiveness* of a graph, explored in the next section, provides a particular ordering of the nodes and hence can significantly reduce the number of colors needed.

4.2.2 Inductiveness of a Constraint Graph

Consider the example in Figure 5. According to Brooks's theorem, this graph is four-colorable; actually, the graph is two-colorable. Here, we examine another graph property which generalizes Brooks's result to give a stronger upper bound on the number of colors. More importantly, the property can be used to order the instances in the training data set to make the feasibility problem easy. The value of q (to be defined) along with this ordering gives us another sufficient condition which if satisfied when adding in new constraints will allow for an efficient incremental clustering algorithm.

The following definition is from [10].

DEFINITION 4.1. *Let q be a positive integer. An undirected graph $G(V, E)$ is q-**inductive** if the nodes of G can be assigned distinct integer values in such a way that each node has an edge to at most q nodes with higher assigned values.*

To illustrate this definition, consider a star graph $G(V, E)$ with n nodes. Let v_0 denote the center node (with degree $n-1$) of the star and let v_i, $1 \leq i \leq n-1$ denote the other $n-1$ nodes (each of which has degree 1). Assign the integer 1 to node v_1, 2 to node v_2, ..., $n-1$ to node v_{n-1} and n to node v_0. This creates the following linear ordering of the nodes: $\langle v_1, v_2, \ldots, v_n, v_0 \rangle$. Examining the nodes in this order, it can be seen that each node has an edge to *at most one* node with a higher assigned value (each node v_i, $i \neq 0$, has one edge to v_0).

The usefulness of q-inductiveness is shown in the following theorem from [10].

THEOREM 4.1. *Suppose $G(V, E)$ is q-inductive. G can be colored using at most $q + 1$ colors.* ■

Thus, the star graph is 1-inductive and is two-colorable, whereas Brooks's theorem would state that it is n colorable. As a further example for the graph in Figure 5, the following is a 1-inductive ordering: *fg*, *abc*, *jk*, *l*, *de* and *hi*; hence the graph is 2-colorable.

The proof of the above theorem actually provides an algorithm that colors the graph G using at most $q+1$ colors. In particular, the algorithm colors the nodes in an order that is the **reverse** of the given q-inductive ordering. This translates into the reasonable heuristic of ordering the instances to cluster from the **most** constrained to the least (if at all) constrained.

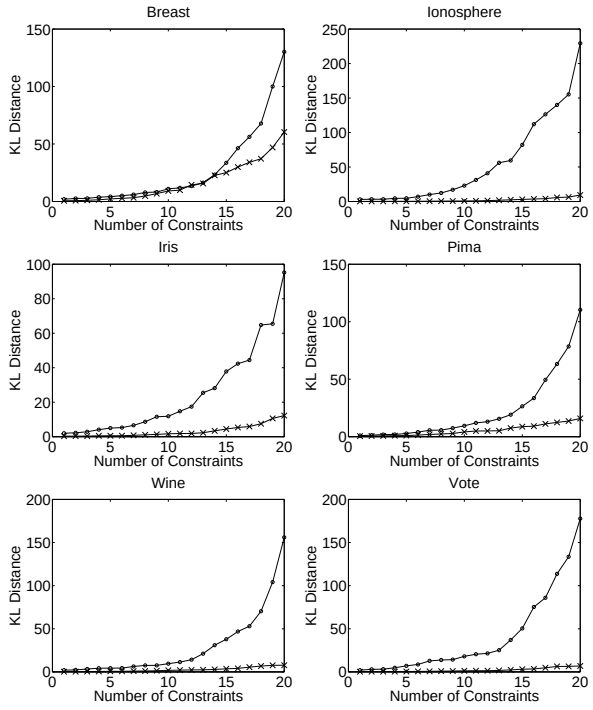
We can make use of existing instance ordering given by the q-inductiveness of a constraint graph to efficiently recolor (re-assign) instances given additional new constraints in linear time to the number of instances to cluster. Indeed the situation will be similar to that of Figure 6. Since the q-inductiveness ordering ensures that the most constrained points are assigned clusters first, there will always be one free color/cluster-id for both x and y.

It should be noted that the instances may need to be reordered when new constraints are added to the old set.

5. PUTTING IT ALL TOGETHER: AN EFFICIENT INCREMENTAL CONSTRAINED CLUSTERING ALGORITHM

It should be noted that the following algorithm is only correct when our basic and non-trivial conditions listed in sections 4.1 and 4.2 occur. The algorithm takes an input a single constraint at a time and depending on the properties of the constraint will attempt to greedily optimize the objective function f. If the constraint does not improve f then the constraint is passed over and the user chooses another. Furthermore, since finding a clustering to satisfy a must-link constraint between two points that are already constrained by cannot-link constraints cannot be done efficiently, we also inform the user of this situation and pass over the constraints. The algorithm is shown in its entirety in Figure 8.

Figure 7: Graph of the Average KL distance (over 500 repetitions) between the initial constrained clustering under 20 randomly chosen constraints and the clustering obtained from these 20 and additional constraints (1 to 20). Series -x- indicates incremental constrained clustering results and series -o- indicates non-incremental total constrained reclustering results. Number of clusters k is set to number of extrinsic classes.



6. EXPERIMENTAL RESULTS

We now present results from testing the algorithm described in Figure 8 under the stronger sufficient condition of $k > q + 1$. Our first set of experimental results shows that the approach of adding in new constraints and then totally reclustering the data (i.e. the non-incremental/total-reclustering approach) using COP-k-means [14] has several significant limitations. However, if there are substantially many more constraints to be added this is a reasonable approach. The second set of experimental results shows that our incremental clustering algorithm is efficient with respect to the number of instances moved and interestingly the resultant clustering quality is typically better than total reclustering of all the available data under the constraints.

6.1 Limitation of Total Re-Clustering

We take six UCI data sets and randomly choose to keep only 10% of the data point labels. From these labeled points we will generate ML constraints (when the labels agree) and CL constraints (when the labels disagree) by randomly choosing two points at a time. We begin by clustering using COP-k-means [14] the various UCI data sets with twenty initial *randomly* chosen constraints (C) to produce the initial clustering Π. The COP-k-means algorithm attempts to find a set partition with minimum vector quantization error that satisfies all of the constraints. We randomly choose more constraints which are then given to our greedy incremental clustering algorithm (Figure 8) one at a time. The algorithm will accept an additional twenty constraints (C'). Note that these twenty additional constraints are actively chosen in the sense that they reduce the objective function f, which in this case is the average label purity over all clusters over all points.

In the non-incremental clustering case, each constraint is added one at a time to C, we recluster the data with the initial twenty and progressively added constraints. We then report the difference between the new clustering and the initial clustering with respect to the Kullback-Leibler (KL) distance of the respective mixture distributions each clustering defines. It should be noted that, for each run, we use the same random number seed and hence the same cluster centroids; so, the only thing that changes from experiment to experiment is the set of additional constraints.

These results, shown in Figure 7, indicate that the approach of total re-clustering can lead to significantly different clusterings compared to the initial clustering, while the results of incremental clustering remain rather similar to the initial clustering even for up to 20 incrementally added constraints. This will often be desirable, since the user may prefer the existing clustering for application-oriented reasons. Adding in a single constraint and reclustering can give vastly different results, particularly if the new constraint contains instances that are towards the start of the ordering of how the instances will be assigned. For a fair comparison to see how different the initial and final clusterings would be, we gave the non-incremental constrained clustering algorithm the constraints chosen by our algorithm. We will not do this for the remainder of our experimental section.

It is tempting to think that since non-incremental clustering can greatly change a clustering solution, then why not just fix all instances **not** involved in the **new** constraints. However, due to the transitivity of ML constraints ($ML(x, y)$, $ML(y, z) \rightarrow ML(x, z)$) and the entailment property of CL constraints ($ML(a, b)$,$ML(c, d)$,$CL(a, c) \rightarrow CL(a, d)$,$CL(b, c)$, $CL(b, d)$) many points not explicitly in C' have additional constraints placed on them and the points that can be fixed may be quite small. Furthermore, fixing points to be in a particular cluster may lead to the situation where a partition that satisfies the joint constraint set ($C \cup C'$) may not exist.

6.2 Benefits of Greedy Incremental Algorithms

In this section we show that incremental clustering as expected is more efficient and also that the results obtained by using an incremental algorithm compare favorably with non-incremental (total-reclustering) COP-k-means.

We repeat a similar set of experiments as before, except that this time we measure how much "work" is performed by each algorithm. For incremental clustering this is measured as the number of instances moved from one cluster to another. For non-incremental clustering it is the number of changed assignments from iteration to iteration for all iterations until the algorithm converges. Results are shown in Figure 9.

When comparing the accuracy of incremental constrained clustering with two styles of non-incremental constrained clustering (COP-k-means and Basu, Bilkeno and Mooney's MKM algorithm [1] that learns a distance function) we see some interesting results. Firstly, when compared to algo-

rithms that attempt to satisfy all constraints (as it does) incremental constrained clustering compares favorably and indeed performs better. This is to be expected as by incrementally specifying the constraints (rather than all at once) the incremental algorithm does not get over-constrained as the non-incremental variant can [7].

However, both non-incremental and incremental algorithms that satisfy all the constraints are typically outperformed by distance learning algorithms. The latter class of algorithms interpret $ML(a, b)$ and $CL(x, y)$ as indicating that a and b should be close together and x and y should be far apart in some learnt distance function. The reason why these approaches perform better than those satisfying all constraints is that learning a distance metric means not only that a and b are closer together (most likely in the same cluster) but all points surrounding a and b are closer together. In this way, each constraints is helpful beyond those points that are part of the constraint. Typical experimental results are shown in Figure 10. In all cases, incremental algorithms that satisfy all constraints perform better than non-incremental algorithms that satisfy all constraints; sometimes, the former class of algorithms perform as well as distance metric learning. This result indicates that investigating the problem of incrementally learning distance functions should be profitable.

7. CONCLUSION AND FUTURE WORK

Previous work by ourselves and others has established the benefits of clustering under a batch of given instance level constraints. In this paper we look at the problem of incremental constrained clustering. We explore several problem definitions that allow the user to provide feedback by critquing an existing clustering through constraints and to receive feedback on how useful the additional constraints were at further optimizing a given objective function. This approach allows for a two-way feedback: The user presents feedback to the algorithm in the form of a constraint. The algorithm provides feedback to the user by indicating whether the given constraint was useful.

However, our complexity analysis show that both adding and removing constraints is typically intractable. In this paper, we focus on the constraint addition problem and show that just adding a *single* constraint (be it ML or CL) is in the worst case intractable. However, we identify two *sufficient* conditions when the feasibility problem for adding of constraints is easy which translate into restrictions on the *types* of constraints that can be added. For example, the first condition, namely Brooks's theorem, requires a user to choose new CL constraints so that no point is part of k or more CL constraints. The second condition, namely Irani's q-inductiveness condition, restricts a user to those CL constraints which ensure that each point x is cannot-linked to at most $k-1$ points that follow x in the chosen ordering.

We then developed an efficient algorithm for these sufficient conditions that incrementally allows feedback and tested it using simulated feedback from small amounts of labeled data. Our results show that a) just adding one constraint but not performing incremental clustering, rather re-running the constrained clustering algorithm using the same random number seed and initial centroids can produce quite different clusterings b) incremental clustering is more efficient than re-running the entire clustering algorithm with the additional constraints and c) incremental clustering performs better than non-incremental clustering because the latter can get over-constrained as reported earlier [7]. However, both the incremental and non-incremental algorithms that attempt to satisfy all constraints can sometimes perform significantly worse than algorithms that learn a distance metric from the constraints and then using it to cluster the data.

For future work we wish to investigate what is the maximum number of additional constraints allowed before complete re-clustering should be performed. Furthermore, for the $k = 2$ case we have preliminary results for efficient algorithms to incrementally add both ML and CL constraints which we wish to further explore. Finally, the problem of incrementally modifying a distance function remains an important open question.

8. ACKNOWLEDGEMENTS

The first author gratefully acknowledges support of this work by NSF grant number 0643668 titled *CAREER: Knowledge Ehanced Clustering using Constraints*.

9. REFERENCES

[1] Basu, S., Bilenko, M., Mooney, R.J., A probabilistic framework for semi-supervised clustering, ACM KDD, 2004.

[2] Bar-Hillel, A., Hertz, T., Shental, N., Weinshall, D., Learning a Mahalanobis metric from equivalence constraints. *JMLR* **6** (2005)

[3] D. Cohn, R. Caruana, and A. McCallum, "Semi-supervised clustering with user feedback", Tech. Report TR2003-1892, Cornell University, 2003.

[4] I. Davidson and S. S. Ravi, "Clustering with Constraints: Feasibility Issues and the k-Means Algorithm",SDM 2005.

[5] I. Davidson and S. S. Ravi, "The Complexity of Non-Hierarchical Clustering with Instance and Cluster Level Constraints", ECML/PKDD 2005.

[6] I. Davidson and S. S. Ravi, "Identifying and Generating Easy Sets of Constraints For Clustering", AAAI 2006.

[7] I. Davidson and S. S. Ravi, "The Complexity of Non-Hierarchical Clustering with Constraints", *Journal of Knowledge Discovery and Data Mining* (DMKD), Volume 14, Number 1, Pages 25-61. 2007.

[8] Davidson I., Wagstaff, K., Basu, Sugato., Measuring Constraint-Set Utility for Partitional Clustering Algorithms, Proceedings of ECML/PKDD 2006.

[9] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman and Co, 1979.

[10] Irani, S. "Coloring Inductive Graphs Online", *Algorithmica*, Vol. 11, No. 1, Jan. 1994, pp. 53–72.

[11] Lu, Z., Leen, T.K., Semi-supervised learning with penalized probabilistic clustering, NIPS 2005.

[12] Rand, W.M., Objective criteria for the evaluation of clustering methods, *Journal of the American Statistical Association* **66**(366), 1971.

[13] D. B. West, *Introduction to Graph Theory*, 2^{nd} Edition, Prentice Hall, Inc., 2001.

[14] K. Wagstaff & C. Cardie. Clustering with Instance-Level Constraints, ICML 2000.

Input: $\Pi = \{S_1 \ldots S_k\}$: a clustering of point set S into k clusters satisfying all the ML & CL constraints in C.
$f(\Pi)$: an objective function of Π.
$f(\Pi, move(x_1, from_1, to_1) : \ldots move(x_m, from_m, to_m))$: an objective function of Π after applying given moves.
r: the number of constraints the user wishes to add.
Output: A partition Π' of S into k clusters with constraints in $C \cup C'$ all satisfied and f is heuristically minimized.

1. z = 1 ; $C' = \emptyset$
2. $I \in \{ML(x,y), CL(x,y)\}$: the new constraint.
3. **if** SufficientCondition($C \cup C' \cup I$) = **false goto** step 2.
4. Let $a = Cluster(x)$ and $b = Cluster(y)$.
5. **switch(I) do**

   $ML(x,y)$, $x, y \notin C \cup C'$ (pure points, see section 4.1)
   if $\nexists\ i$:$x, y \in S_i$ **then**
   Π' = Modify Π by moving x, y to j : $argmin_j f(\Pi, move(x,a,j) : move(y,b,j))$
   if $f(\Pi') < f(\Pi)$ **then** $z = z + 1$; $C' = C' \cup I$;
   if $z > r$ **then** exit.

   $CL(x,y)$, $x, y \notin C \cup C'$ (pure points, see section 4.1)
   if $\exists\ i$: $x, y \in S_i$ **then**
   Π' = Modify Π by moving x, y to clusters i, j : $argmin_{i,j} f(\Pi, move(x,a,i) : move(y,b,j))$
   if $f(\Pi') < f(\Pi)$ **then** $z = z + 1$; $C' = C' \cup I$;
   if $z > r$ **then** exit.

   $ML(x,y)$, $x \notin C \cup C', y \in C \cup C'$ (x is pure, see section 4.1)
   if $\nexists\ i$:$x, y \in S_i$ **then**
   Π' = Modify Π by moving x to Cluster b
   if $f(\Pi') < f(\Pi)$ **then** $z = z + 1$; $C' = C' \cup I$;
   if $z > r$ **then** exit.

   $CL(x,y)$, $x \notin C \cup C', y \in C \cup C'$ (x is pure, see section 4.1)
   if $\exists\ i$: $x, y \in S_i$ **then**
   Π' = Modify Π by moving x to cluster j: $argmin_j f(\Pi, move(x,a,j))$
   if $f(\Pi') < f(\Pi)$ **then** $z = z + 1$; $C' = C' \cup I$;
   if $z > r$ **then** exit.

   $ML(x,y)$, $x, y \in C \cup C'$ (x, y are already constrained, see section 4.2)
   if $\nexists\ i$: $x, y \in S_i$ **then**
   return **no efficient series of moves.**

   $CL(x,y)$, $x, y \in C \cup C'$ (x, y are already constrained, see section 4.2)
   if $\exists\ i$: $x, y \in S_i$ **then**
   Let the set Q be clusters not cannot-linked to x.
   Let the set R be clusters not cannot-linked to y.
   Π' = Modify Π by moving x to cluster i and y to cluster j: $argmin_{i \in Q, j \in R, i \neq j} f(\Pi, move(x,a,i) : move(y,b,j))$.
   if $f(\Pi') < f(\Pi)$ **then** $z = z + 1$; $C' = C' \cup I$;
   if $z > r$ **then** exit.

6. $\Pi = \Pi'$; **goto** step 2.

Figure 8: Efficient Incremental Algorithm

Figure 9: Graph of the Average Number of Moves (over 500 repetitions) required to find a solution to satisfy a varying number of additional constraints (1 to 20) to the initial 20 constraints. Series -x- indicates incremental constrained clustering results and series -o- indicates non-incremental total reclustering results. Number of clusters k is set to number of extrinsic classes.

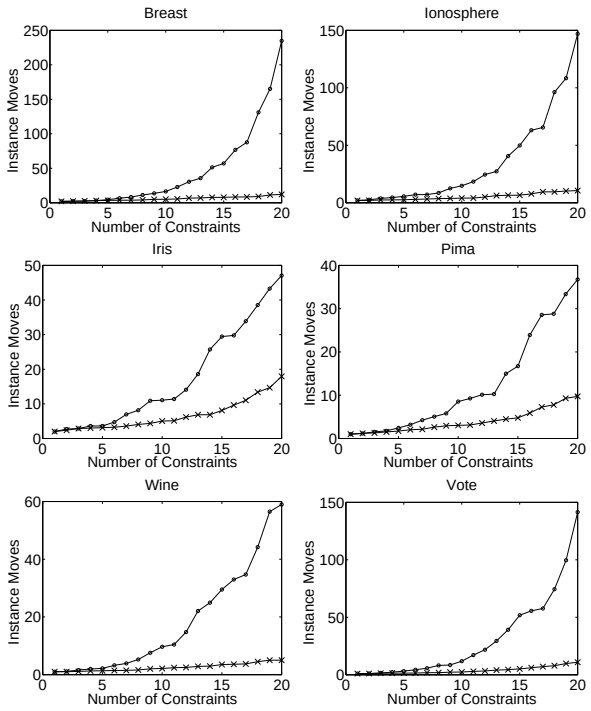


Figure 10: Three typical results for average accuracy (over 500 repetitions) as measured using the cluster purity. Series -x- indicates incremental constrained clustering results and series -o- indicates non-incremental total reclustering results with COP-k-means and the solid line indicates non-incremental total reclustering with Basu, Bilenko and Mooney's distance learning technique MKM. Number of clusters k is set to number of extrinsic classes.