

# Identifying and Generating Easy Sets of Constraints For Clustering

Ian Davidson and S. S. Ravi

Department of Computer Science, University at Albany - SUNY, 1400 Washington Avenue, Albany, NY 12222

Email: {davidson, ravi}@cs.albany.edu

## Abstract

Clustering under constraints is a recent innovation in the artificial intelligence community that has yielded significant practical benefit. However, recent work has shown that for some negative forms of constraints the associated sub-problem of just finding a feasible clustering is **NP**-complete. These *worst case* results for the entire problem class say nothing of where and how prevalent easy problem instances are. In this work, we show that there are large pockets within these problem classes where clustering under constraints is easy and that using easy sets of constraints yields better empirical results. We then illustrate several sufficient conditions from graph theory to identify *a priori* where these easy problem instances are and present algorithms to create large and easy to satisfy constraint sets.

## Introduction and Motivation

Clustering is a ubiquitous unsupervised learning activity used within artificial intelligence for object identification in images, information retrieval and natural language understanding (Wagstaff et' al. 2001). A recent innovation has been the introduction of clustering under instance-level constraints which effectively provide hints to the composition of a desirable clustering of the instances. The most prevalent form of constraints are must-link (ML) where two instance must be in the same cluster and cannot-link (CL) where they are in different clusters. These two types of constraints offer the ability to incorporate strong background knowledge into the clustering process. For example when clustering automobile GPS trace information to form clusters that are traffic lanes (Wagstaff et' al. 2001), the physical distance between lanes (4 meters) can be used to generate CL constraints between instances. In practice constraints are typically randomly generated from small amounts of labeled data. If two randomly chosen instances have the same (different) label an ML (a CL) constraint is generated between them.

The main uses of ML and CL constraints have been in the context of three classes of algorithms listed below. a) Algorithms that satisfy every constraint, such as COP- $k$ -means (Wagstaff et' al. 2001), b) algorithms that satisfy as many constraints as possible, such as PKM (Bilenko et' al. 2004), and c) algorithms that learn a distance function (Xing et' al. 2003) so that points which are part of an ML (a CL)

constraint are close (far) according to the distance function. Some algorithms have multiple objectives, such as MPKM (Bilenko et' al. 2004) which attempts both b) and c).

However, when clustering under constraints, the feasibility sub-problem arises: Does there exist *any* solution that satisfies all constraints? For example, there is no possible clustering under the constraints  $ML(a, b)$ ,  $CL(a, b)$ , but even for self-consistent constraints set such as  $CL(a, b)$ ,  $CL(b, c)$  and  $CL(a, c)$ , there is no clustering for  $k \leq 2$ . Formally:

**Definition 1. The Feasibility Problem.** Given a set  $D$  of data points, a collection  $C$  of ML and CL constraints on some points in  $D$ , upper ( $K_u$ ) and lower bounds ( $K_l$ ) on the number of clusters, does there exist at least one partition of  $D$  into  $k$  clusters such that  $K_l \leq k \leq K_u$  and all the constraints in  $C$  are satisfied?

If this question can be answered efficiently, then one can generate feasible clustering at each iteration of a clustering under constraints algorithm. Previous work (Davidson & Ravi, 2005a) has produced *worst case* results for the feasibility problem for clustering under ML and CL constraints amongst others. The feasibility problem for clustering under ML constraints is in **P** while clustering under CL only and ML and CL is **NP**-complete, as can be shown by a reduction from graph coloring. Because of this complexity result, it is tempting to abandon the use of CL constraints, however, the transitivity and entailment property of ML and CL constraints respectively (Wagstaff et' al. 2001) can result in many entailed CL constraints and hence are quite useful. For example, the constraints  $ML(a, b)$ ,  $ML(a, c)$ ,  $ML(d, e)$ ,  $CL(a, e)$  entail the additional constraints  $ML(a, c)$ ,  $CL(a, d)$ ,  $CL(b, d)$ ,  $CL(b, e)$ ,  $CL(c, d)$  and  $CL(c, e)$ .

The worst case complexity results just make a general statement about the feasibility problem, namely that it contains a **core** of difficult problem instances. Of more practical importance to users and A.I. researchers are pragmatic questions such as how does one *a priori* identify easy problem instances and how does one generate them? How we define an *easy* problem instance is important. An idealized definition would of course be a set of constraints where there is at least one feasible solution. However, since the feasibility problem is **NP**-complete for CL constraints, testing any necessary and sufficient condition to *a priori* identify feasible constraint sets cannot be carried out efficiently so long

as  $\mathbf{P} \neq \mathbf{NP}$ . Furthermore, knowing that a constraint set has a feasible solution does not make it easy to find the solution. As we shall see later, the ordering of the instances in  $D$  plays an important role for most iterative style clustering algorithms such as  $k$ -means and EM which typically assign instances to clusters in a fixed pre-determined order. Therefore, we will adopt the definition that a problem instance is easy for a clustering algorithm if a feasible solution can be found given an *ordering* of the instances in the data set which in turn determine the order of how the instances will be assigned clusters. Formally:

**Definition 2.** A feasibility problem instance is  $\beta$ -easy if a feasible solution can be found by an algorithm  $\beta$  given the ordering of instances in the training set which determines the order they will be assigned to clusters.

It may help the reader to loosely interpret an easy problem as being under-constrained and a hard problem as being over-constrained such that a feasible solution is difficult to find. In this paper we shall focus on using Definition 2 with respect to the simplest constrained clustering algorithm, namely COP- $k$ -means (Wagstaff et al. 2001). We expect constraint sets which are easy for COP- $k$ -means are also easy for other more complex algorithms that assign instances to clusters in a fixed order.

This paper attempts to answer the question of what properties identify easy problems *a priori* and how one can generate them in a manner that is useful to practitioners and algorithm designers. We begin by summarizing the previous worst case results for the feasibility problem (Davidson & Ravi, 2005a). We then empirically investigate properties in the feasibility landscape showing that there are large parts which are easy. We also empirically illustrate that using easy sets of constraints affects the performance of a wide variety of algorithms that use constraints. We then discuss two sufficient conditions to identify easy feasibility problems and then develop an algorithm to generate large constraint sets and an ordering of the instances that makes the feasibility problem easy (see Definition 2).

Our work is different from that of active constraint generation (Basu et al. 2004) since that work assumes access to a noiseless Oracle that knows the labels of all instances whereas we assume that only a small amount of labeled data is available.

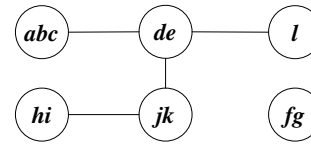
## A Graph Coloring Interpretation of Clustering Under Constraints and Worst Case Results

Consider the problem of clustering under an expanded version of the set of constraints described earlier:  $ML(a, b), ML(a, c), ML(d, e), ML(f, g), ML(h, i), ML(j, k), CL(a, e), CL(i, j), CL(d, k), CL(e, l)$ .

Regardless of the total number of instances, the feasibility problem is limited to instances  $a, b, \dots, l$ . The following construction represents this problem as a graph:

1. Compute the transitive closure of the must-linked constraints, producing connected components  $C_1, \dots, C_r$ .
2. Create one node for each connected component and for each instance that is part of only CL constraints.

Figure 1: A graphical representation of the feasibility problem for  $ML(a, b), ML(a, c), ML(d, e), ML(f, g), ML(h, i), ML(j, k), CL(a, e), CL(i, j), CL(d, k), CL(e, l)$



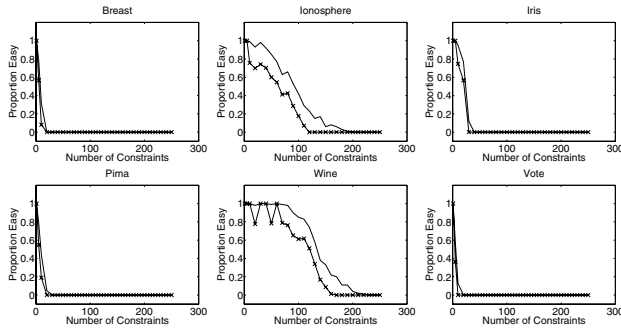
3. Place an edge between those nodes that represent instances that are part of CL constraints.

Figure 1 shows the result for our simple example. Finding a feasible solution to satisfy all of the must-link constraints, simply involves computing the transitive closure in step 1) above and checking whether the number of connected components is at least  $K_\ell$ . Since the transitive closure computation takes time  $O(n + m)$ , where  $n$  is the number of nodes and  $m$  the number of edges, the feasibility problem for ML constraints is in  $\mathbf{P}$ . Clustering to satisfy the CL constraints is then a case of assigning each node an integer value from 1 to  $k$  so that no two adjacent nodes have the same value. This is at least as hard as the graph  $k$ -coloring problem which is known to be  $\mathbf{NP}$ -complete. To find a feasible clustering in our example we can assign instances  $f$  and  $g$  to any cluster so long as they are together. However, care must be taken to assign the instances  $de$  so that the assigned value does not conflict with the assignments for  $l$  and  $jk$  and  $abc$ . We shall see later on that the node in the graph (namely,  $de$ ) with the maximum degree (namely, 3) has special significance in determining easy problem instances as does the ordering of the nodes used to assign colors. Reference (Davidson & Ravi, 2005a) presents complete complexity proofs.

## Investigating the Difficulty Landscape

The previous section's worst case complexity results stated the clustering under CL or ML and CL constraints is, in general, difficult. In this section, we shall empirically explore the difficulty landscape to identify the prevalence of easy problem as per Definition 2 using a popular clustering under constraints algorithm, COP- $k$ -Means (Wagstaff et al. 2001). This algorithm is a common method of extending  $k$ -Means to ensure that all ML and CL constraints are satisfied at each iteration. It is important to note that the COP- $k$ -Means algorithm is sensitive to the ordering of the points to be clustered; therefore, if it does converge for a set of constraints, then the constraint set can be deemed easy with respect to given the ordering of the points. The COP- $k$ -Means algorithm handles ML constraints by first computing the transitive closure and then replacing all the points in each transitive closure set with a single point which is the centroid of all the points in that set, weighted by the size of the connected component. Since  $k$  is typically small, infeasibility due to ML constraints does not occur. The algorithm incorporates CL constraints by changing nearest-centroid assignment to *nearest-feasible-centroid* assignment. This is easily programmed by creating for each point a sorted list of centroids (in increasing order of distances) and progressing down the list until a feasible centroid is found to assign the

Figure 2: Graph of the proportion of the 500 randomly created sets of CL constraints that are easy (Solid Line) according to Definition 2 and (Crossed Line) our sufficient condition:  $k > 1 + \text{MaxDegree}(\text{CLGraph})$ .



point (Wagstaff et’ al. 2001) without violating a CL constraint. For any point, if the algorithm reaches the bottom of this list without finding a feasible assignment, the algorithm halts with an indication that it cannot find a feasible solution.

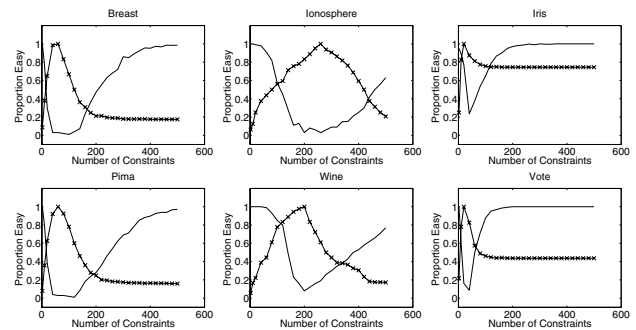
In all our experiments, we use the following common method for generating ML and CL constraints. If the labels of the two points agree, then an ML constraint is generated; otherwise, a CL constraint is generated. We generate a variety of constraint set sizes 500 times and then determine what proportion of them are easy according to the above definition. The instance labels are of course not given to the clustering algorithm and we cluster with  $k$  equaling the number of extrinsic classes in the UCI data sets. We set aside 20% of data points as labeled from which we generate constraints; the remaining 80% of the points are unlabeled.

With only CL-constraints (Figure 2) we see that for the solid line as the number of constraints increases, the proportion of easy constraint sets decreases and we also note that the transition point from being easy to not-easy is abrupt and is at different position for each data set. The crossed line is the proportion of times our yet to be identified sufficient condition occurs. With ML and CL constraints, we note a quite unusual phenomenon shown in Figure 3 (solid line). This result is quite counterintuitive. Initially for a small number of constraints the feasibility problem is easy for the vast majority of the 500 trials. However, the number of easy constraint sets diminishes as the number of constraints increases only to become more prevalent again! In the experiments used to construct Figure 3, the numbers of ML and CL constraints are equal.

### Why Problems are Difficult

It is of course interesting to investigate the question why some problem instances are easy for COP- $k$ -means and others are difficult. There are two potential reasons. Firstly, the problem may have no feasible solution such as clustering a dataset containing the constraints  $CL(a, b), CL(b, c), CL(a, c)$  with  $k = 2$ . Secondly, the ordering of instances in the training set can make finding the solution difficult. Once again consider Figure 1. The graph is 2 colorable; however, suppose the nodes are considered in the order  $\langle abc, hi, de, jk, l, fg \rangle$ . The algorithm may as-

Figure 3: Graph of the proportion of the 500 randomly created sets of ML and CL constraints that are easy (Solid Line) according to Definition 2 and (Crossed Line) the average number of connected components (rescaled to fit on the same axes).



sign color 1 to both nodes  $abc$  and  $hi$  and color 2 to node  $de$ . Now, when node  $jk$  is considered, it cannot be assigned either color 1 or color 2 without violating a CL constraint. Thus, the algorithm may terminate with an indication that no feasible solution exists. Since all the non-hierarchical clustering algorithms we empirically investigate process instances in a fixed ordering, they may not converge or converge to a poor solution depending on the instance ordering.

### Effects of Using Easy Sets of Constraints

Previous sections defined feasibility and showed that easy sets of constraints occur frequently for some constraint set sizes despite worst case complexity results. But is there any benefit to identifying and using easy sets of constraints? This section answers this question empirically.

The effect of using an easy set of constraints on algorithms that attempt to satisfy all constraints is clear: they will converge to a feasible solution. However, as mentioned in the introductory section, there are other algorithms that make use of constraints. In this section we shall explore the effect of COP- $k$ -means-easy sets of constraints on a) the PKM (partial constraint satisfaction  $k$ -means) algorithm that attempts to satisfy as many constraints as possible (Bilenko et’ al. 2004), b) the MPKM (metric learning partial constraint satisfaction  $k$ -means) algorithm that learns a distance function and attempts to satisfy as many constraints as possible (Bilenko et’ al. 2004) and c) a simple closest-cluster-join agglomerative hierarchical clustering (CHC) algorithm that attempts to satisfy all constraints and terminates building the dendrogram once it cannot (Davidson & Ravi, 2005b).

Firstly, we explore algorithms that learn distance functions and perform clustering when constraints can be ignored. We present the results for all six data sets for when the number of constraints sets that are easy or not-easy are approximately the same (i.e. number of constraints in Figure 2 when y-axis equals approximately 0.5). For each data set and non-hierarchical clustering technique we present the number of iterations until convergence and error of the algorithm as measured on the extrinsic class label for *all* of the instances, not just those used to generate constraints. We see (Table 1) that for both algorithms, for easy constraint sets,

Dataset-Technique	Difficult - Easy Iterations	Difficult - Easy Error
Breast-MPKM	2.21 (0.43)	3.06 (1.77)
Breast-PKM	2.89 (0.78)	4.22 (2.68)
Ion.-MPKM	2.08 (1.33)	1.66 (0.99)
Ion.-PKM	2.31 (1.78)	2.36 (1.43)
Iris-MPKM	0.41 (0.23)	1.21 (0.60)
Iris-PKM	0.12 (0.04)	0.84 (0.57)
Pima-MPKM	0.47 (0.41)	1.96 (1.42)
Pima-PKM	0.47 (0.40)	1.97 (1.56)
Wine-MPKM	3.26 (1.09)	2.01 (1.17)
Wine-PKM	1.87 (1.04)	2.31 (1.69)
Vote-PKM	1.01 (0.86)	1.37 (0.78)
Vote-MPKM	1.22 (1.01)	1.42 (1.11)

Table 1: Mean difference in performance of various techniques on standard UCI data sets for approximately equal number of easy and non-easy sets of CL constraints. Results are averaged over 500 constraint sets. P values are shown within parenthesis at 99% confidence level.

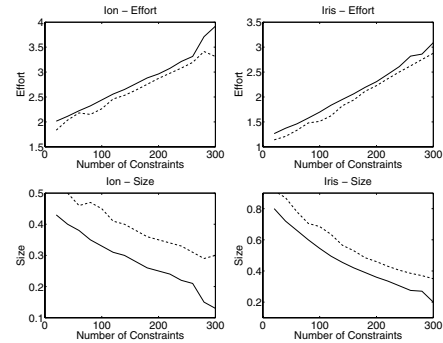
the reduction in number of iterations and error is significant at the 99% confidence level and in particular, the error rate improvement is practically significant.

Due to space reasons Figure 4 shows the results for the agglomerative hierarchical clustering algorithm CHC for only two of the data sets; similar results hold for the remaining four data sets. The first row of graphs illustrate the effort (join attempts) required to build a dendrogram as a multiple of the number of initial clusters less one. Since at each level of the dendrogram a join must be between two clusters that will not violate any constraint, more potential join calculations are required than for unconstrained dendrogram construction (just the number of initial clusters less one). We see that on average whenever the constraint set is easy results in less computational effort regardless of the number of constraints. Similarly, the second row of graphs show how large the dendrogram that can be constructed as a fraction of the unconstrained dendrogram height. We see again that when the constraint set is **not** easy, the dendrogram prematurely terminates compared to when the constraint set is easy. The notion of premature termination of dendrogram construction is essentially that the constraint set along with the distance measure causes a series of joins that dead-ends at a dendrogram height that is sub-optimal in that another series of joins could have built a taller/fuller dendrogram. The notion of premature termination for hierarchical clustering under constraints is discussed extensively in (Davidson & Ravi, 2005b).

### Sufficient Conditions for Easiness

Our previous sections have shown that there are easy constraint sets and that using easy constraint sets leads to better empirical results. In this section we outline two properties of the constraint graph (see Figure 1 for an example) that are sufficient (but not necessary) for identifying *COP-k-means*-easy constraint sets. The first condition is a set theoretic property of the constraints regardless of the data set order-

Figure 4: Graph of the mean effort for the CHC algorithm to build a dendrogram and its size over 500 randomly generated ML and CL constraint sets. The solid line is for difficult problems, dashed line for easy problems.



ing. The second incorporates the data set ordering. In a later section we illustrate how to use these conditions to generate a set of constraints that are easy to satisfy.

### Brooks's Theorem

To explain the phenomenon observed in Figures 2 and 3, we will use Brooks's theorem (West, 2001) and the notion of *entailed constraints*. Brooks's theorem points out that if the maximum node degree of a graph  $G$  is  $\Delta$  and  $k \geq \Delta + 1$ , then the  $k$ -coloring problem is easy; in particular, graph  $G$  can always be colored using  $k$  colors. Such a coloring can be obtained using *any* linear ordering of the nodes. An example of such an algorithm is to assign to the current node  $v$  a color that has not been used for any of the neighbors of  $v$  which have already been colored.

The COP- $k$ -means style of clustering under constraints can be viewed as a greedy type of graph coloring to find a feasible solution that also minimizes the algorithm's objective function. The centroids tessellate the instance space into regions and the COP- $k$ -means strategy is to assign each node a color according to this tessellation. Brooks's theorem gives us a sufficient condition to identify when greedy coloring algorithms will always converge and when they won't. In easy instances of the feasibility problem, the value of  $\Delta$  (the maximum degree of the constraint graph) is less than  $k$ , and the coloring algorithm based on Brooks's theorem will *always* succeed. Informally, if the maximum number of CL-constraints involving the same point is at most  $k - 1$ , then the feasibility problem is easy; otherwise, the feasibility problem will be difficult (i.e., the nearest-feasible-centroid assignment approach will fail in general). As the number of constraints increases, the proportion of easy constraint sets diminishes since the chance of a node being encountered in  $k$  or more constraints increases. In figure 2 the crossed-line plots the proportion of times the maximum degree of the graph is less than  $k$ . Since Brooks's theorem is a *sufficient* condition for easy problems, the occurrence of the sufficient condition always under-estimates how often the problem is easy in practice.

Explaining the results for both ML and CL constraints requires more effort and the use of the transitivity and entailment properties of ML and CL constraints respectively. When the number of constraints is small, the number of con-

nected components in the graph representing the clustering problem would also be small. So, the chance of any node being part of more than  $k$  CL constraints is small. Thus, the problem is easy. Conversely, when the number of constraints is large, the number of connected components will be typically the number of extrinsic labels (due to the transitive nature of ML-constraints), and hence the maximum degree of the graph is  $k - 1$  (i.e., there is a CL-constraint between each pair of connected components). Thus, the maximum node degree  $\Delta$  and  $k$  satisfy the condition  $k \geq \Delta + 1$ , and again the problem is easy. However, when the number of constraints is not too large, the number of connected components is typically much greater than  $k$ . This is because many points with the same extrinsic label have not had the opportunity to combine and form a single connected component. Furthermore, the chance of a CL-constraint between two connected components is large due to the entailed constraint property. Figure 3 (crossed-line) shows the number of connected components (normalized to fit on the same axes) which is approximately maximum when the constraint sets are difficult.

The above discussion points out at least two ways of making the constrained clustering problem easy. Firstly, we can make a point part of only one ML or CL constraint. This, is achieved by sampling the points *without* replacement while generating constraints. Alternatively, we can make  $k$  larger than  $\Delta$ . However, these are less than desirable approaches since they limit how we can use our clustering algorithm and constraints. Instead we may generate constraint sets so that the maximum degree of the constraint graph is less than  $k$ . This is achieved by ensuring that each instance is part of  $k - 1$  or fewer constraints. Note that by Brooks's theorem, a  $\Delta + 1$  coloring is possible regardless of the order in which nodes are considered. The notion of *inductiveness* of a graph, explored in the next section, provides a particular ordering which can significantly reduce the number of colors used.

### Inductiveness of Graphs

When the maximum node degree of a graph is  $\Delta$ , Brooks's theorem shows how the graph can be colored using at most  $\Delta + 1$  colors. However, there may be ways to color the graph using far fewer colors. Consider the example in Figure 1. 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 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 following definition is from (Irani, 1994).

**Definition 3.** *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  $v_2, \dots, n - 1$  to node  $v_{n-1}$  and  $n$  to node  $v_0$ . This creates the following linear ordering of the nodes:  $\langle v_1, v_2, \dots, v_n, v_0 \rangle$ . 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  $v_0$ ). Thus, the star graph is 1-inductive.

The usefulness of  $q$ -inductiveness is shown in the following theorem from (Irani, 1994).

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

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.

The advantage of Theorem 1 over Brooks's theorem is that many graphs which have nodes of high degree, are  $q$ -inductive for a small integer  $q$ . For example, regardless of node degrees, it is known that all trees are 1-inductive and that all planar graphs are 5-inductive (Irani, 1994).

We shall use the notion of  $q$ -inductiveness to present an ordering of the training instances so that satisfying all constraints can be easily obtained by processing the instances in that particular ordering. One approach to obtain such an inductive ordering of the nodes is shown in Figure 5. The idea is to repeatedly choose a node of minimum degree from  $G$ , output that node and delete it from the graph. (The degrees of nodes must be updated after each deletion.) Suppose during this iterative process, we keep track of the degrees of the nodes chosen for deletion at each stage, and  $q$  denotes the maximum of these degrees. It can be verified that the resulting ordering is a  $q$ -inductive ordering of the nodes of  $G$ . As an example, for the graph in Figure 1, the following is a 1-inductive ordering:  $\langle fg, abc, l, de, jk, hi \rangle$ .

In Figure 5, the insertion of each node (Step 3(b)) is at the end of the list. Hence, nodes of small degree (least constrained) generally appear at the beginning and nodes with higher degrees (most constrained) appear towards the end. The nodes/instances are assigned to clusters in the **reverse** order of this list (high degree to low degree).

In the worst-case, the algorithm in Figure 5 will output an ordering for which  $q = \Delta$ , the maximum node degree. However, for special classes of graphs, the algorithm will do much better. For example, it can be seen that for trees the algorithm will find a 1-inductive ordering. Likewise, the algorithm will output a 5-inductive ordering for planar graphs.

### Generating Easy Sets of Constraints

One obvious method of using our sufficient conditions is to generate a set of constraints from the labeled data randomly as before and test whether the set is easy. If not, then generating another set of constraints is suggested. In this section we shall explore approaches that use our sufficient conditions to generate large and easy to satisfy sets of constraints.

We begin this section by using the result of Brooks's theorem and *attempt* to generate constraint graphs in which each node has a degree of at most  $k - 1$ . In essence, our constraint generation approach tries to distribute the number of CL constraints amongst as many instances as pos-

---

**Input:** Undirected graph  $G(V, E)$ .

**Output:** An ordering  $\langle v_{i_1}, v_{i_2}, \dots, v_{i_n} \rangle$  of the nodes of  $G$  such that  $G$  is  $q$ -inductive for with respect to this ordering.

1. Set list  $L$  to empty. ( $L$  will give the node ordering.)
2. For each node  $v \in V$ , find the degree of  $v$  in  $G$ .
3. **while**  $V \neq \emptyset$  **do**
  - (a) Find a node  $x \in V$  of minimum degree.
  - (b) Insert  $x$  at the end of list  $L$ .
  - (c) Delete node  $x$  from  $V$  and all nodes incident to  $x$  from  $E$ .
  - (d) For each node  $y$  which had an edge to  $x$ , decrement the degree of  $y$  by 1.
4. Output the nodes in the (left to right) order given by  $L$ .

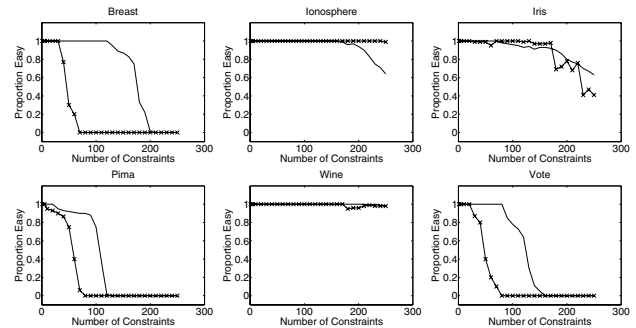
Figure 5: An Algorithm for Inductive Ordering of Nodes

---

sible so that no node is involved in significantly more CL constraints than any other. We then ran the previous experiments reported in Figure 2 except using our new constraint generation approach. The results for this new experiment are shown in Figure 6 crossed line. Comparing these results to Figure 2 we see that we can generate constraint sets with nearly twice as many constraints before the problem becomes over-constrained. This has significant practical benefit as it has been previously shown that more constraints lead to improved clustering quality (Wagstaff et’ al. 2001; Basu et’ al. 2004; Davidson & Ravi, 2005a). The results from the use of Brooks’s theorem seem almost too good to believe. However, upon careful examination of the labeled data the results hold up to scrutiny. Take for example, the Iris data set which contains 150 instances with 50 each of one of the three extrinsic classes. Clustering with  $k = 3$ , it is possible to purposefully generate 75 constraints and create a constraint graph with a degree of at most one and 150 constraints with a maximum degree at most 2. Above 150 constraints the maximum degree of the graph must be three and the problem becomes difficult. Looking at our plot for Iris in Figure 6, it is around the 150 constraint mark that the problem starts to become difficult.

Next we use the algorithm in Figure 5 to reorder constrained instances to form an easy constraint set. Note that this algorithm may create a different instance ordering for each constraint set. We then ran the previous experiments reported in Figure 2 with the *exact same constraint sets* except that the instances were re-ordered in the reverse order of list produced by the algorithm. The results for this new experiment but with reordered instances is shown in Figure 6 (solid line). We see the transition point from when randomly generated constraint sets become consistently (proportion close to 0) difficult has shifted approximately 100 constraints to the right. This effectively means this approach of generating easy constraint sets allows using nearly ten times as many CL constraints before over-constraining the problem.

Figure 6: For training sets reordered according to the  $q$ -inductiveness of the graph (solid line) and constraint sets that attempt to minimize the maximum degree of the constraint graph (crossed line): the proportion of the 500 constraint sets that are easy. Compare with Figure 2.



## Conclusion and Future Work

We have shown that despite worst case results, the difficulty landscape for randomly generated sets of constraints contains many easy (under-constrained) problem instances. Furthermore, our empirical results show that for equally sized easy and difficult constraint sets, the easy sets produce significantly better performance results. We then presented two sufficient conditions to identify easy constraint sets and approaches to create constraint sets that attempt to satisfy the conditions. Our empirical results indicate that creating constraint sets in this purposeful manner (rather than randomly) can lead to using up to ten times as many constraints before the problems becomes difficult (over-constrained). The UCI data sets we used most likely contained some label noise which would cause noisy constraints. Systematically studying our algorithm’s performance using data sets with significant noise is an interesting research question.

## References

- Basu, S., Banerjee, A., & Mooney, R. J. (2004). Active Semi-Supervision for Pairwise Constrained Clustering. *Proc. of 4<sup>th</sup> SIAM Intl. Conf. on Data Mining*.
- Bilenko, M., Basu, S., & Mooney, R. J. (2004). Integrating constraints and metric learning in semi-supervised clustering. *Proc. of 21<sup>st</sup> Intl. Conf. on Machine Learning*.
- Davidson, I., & Ravi, S. S. (2005a). Clustering with constraints: Feasibility issues and the  $k$ -means algorithm. *Proc. of 6<sup>th</sup> SIAM Intl. Conf. on Data Mining*.
- Davidson, I., & Ravi, S. S. (2005b). Hierarchical clustering with constraints: Theory and practice. *ECML/PKDD 2005*.
- Irani, S. “Coloring Inductive Graphs Online”, *Algorithmica*, Vol. 11, No. 1, Jan. 1994, pp. 53–72.
- Wagstaff, K., Cardie, C., Rogers, S., & Schroedl, S. (2001). Constrained  $k$ -means clustering with background knowledge. *Proc. of 18<sup>th</sup> Intl. Conf. on Machine Learning*.
- West, D. B. *Introduction to Graph Theory*, Second Edition, Prentice Hall, Inc., Englewood Cliffs, NJ, 2001.
- Xing, E. P., Ng, A. Y., Jordan, M. I., & Russell, S. (2003). Distance metric learning, with application to clustering with side-information. *NIPS 15*.