Two Approaches To Understanding When Constraints Help Clustering

Ian Davidson
Computer Science
University of California - Davis
davidson@cs.ucdavis.edu

ABSTRACT

Most algorithm work in data mining focuses on designing algorithms to address a discovery problem. Here we focus our attention on designing algorithms to determine the ease or difficulty of a problem instance. The area of clustering under constraints has recently received much attention in the data mining community. We can view the constraints as restricting (either directly or indirectly) the search space of a clustering algorithm to just **feasible** clusterings. However, to our knowledge no work explores methods to count the feasible clusterings or other measures of difficulty nor the importance of these measures. We present two approaches to efficiently characterize the difficulty of satisfying must-link (ML) and cannot-link (CL) constraints: calculating the fractional chromatic polynomial of the constraint graph using Linear Programming (LP) and approximately counting the number of feasible clusterings using Markov Chain Monte Carlo (MCMC) samplers. We show that these measures are correlated to the classical performance measures of constrained clustering algorithms. From these insights and our algorithms we construct new methods of generating and pruning constraints and empirically demonstrate their usefulness.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications-Data Mining

Keywords

Clustering, constraints, semi-supervised learning

1. INTRODUCTION AND MOTIVATION

The last decade has seen extensive work on incorporating instance-level constraints into a variety of clustering and metric learning methods [2]. The two typical instance level constraints are must-link (ML) where two points must be in the same cluster and cannot-link (CL) where the points must be in different clusters. The use of constraints *restricts* the clustering algorithm to explore only those clusterings consistent with users expectations (as given by the constraints). All possible clusterings are then effectively divided into those that are **feasible** (i.e. satisfy all constraints) and those that are **infeasible**. Some algorithms such as COP-k-means algorithm [13] only search the set of feasible clustering while others attempt to satisfy all or most constraints while other still learn a distance function to be used for clustering so that ML (CL) constrained points are close together (far apart).

Pragmatically these constraints can be generated from small amounts of labeled data by generating CL constraints between points with different labels and ML constraints between points with the same label. Alternatively they can be used to enforce geometric properties on the clusters [13]. While most previous work showed that as the size of constraint set becomes larger, the accuracy **on average** of the clustering algorithms increased, a negative result [5] showed that many constraint sets produced a **decrease** in accuracy compared to using no constraints. A valid question is then what differentiates those constraint sets that help constrained clustering versus those that do not. Our work provides several insights:

- We show that the number of feasible clusterings (NFC) and fractional chromatic number (FCN) of the constraints represented as a graph varies greatly for constraint sets of the same size generated using the exact same mechanism.
- The NFC and FCN are highly correlated to the clustering algorithms performance. The former is a measure of the size of the search space, the later the difficulty of generating a legal clustering and transitioning from one state to another.
- Existing constrained clustering algorithms work best when the NFC is large and FCN is small (see Figures 3 and 4).
- The above insights and computational methods can be used to generate new constraint generating and pruning mechanisms to maximize algorithm performance (see section 6 and Figures 5, 6 and 7).

2. BACKGROUND

Given a set X of n points to cluster, the number of possible *unique* clusterings into k groups is given by a Stirling number of the second kind written as $S(n,k)$. However, most clustering algorithms allow both $\{[a,b],[c,d]\}$ and $\{[c,d],[a,b]\}$

KDD'12, August 12–16, 2012, Beijing, China.

to be two different and distinct clusterings since though a, b are together in both clusterings, they are assigned to different cluster numbers. Therefore, the size of the search space of non-hierarchical clustering algorithms is $NF_\emptyset = k!S(n,k)$ clusterings where $S(n,k) = \frac{1}{k!}\sum_{i=0}^{k}(-1)^i\binom{k}{i}(k-i)^n$. NF is our ongoing notation meaning the **N**umber of **F**easible clusterings under the constraint set given by the subscript which is in this case, the empty set. *For the remainder of this paper when we refer to the number of feasible clusterings, we shall effectively refer to the number of feasible clustering of* ***constrained*** *points.* We can easily determine the number of feasible clusterings of all points from the number of feasible clusterings of just constrained points.

Constraint Graphs. Here we show how to convert constraints into a graphical form. To understand the easiness and difficulty of clustering with constraints consider the following problem of clustering under the set of constraints: *C={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)}*. We can represent the problem of finding a single feasible clustering in graphical form as shown in Figure 1 where must-linked points are represented by the same node and cannot-linked points have an edge between them. These nodes and edges form the constraint-graph and to find a feasible clustering is then to assign each node a value between 1 through k so that no two nodes with a common edge have the same value. This is equivalent to the graph coloring problem and forms the basis of the **NP**-Complete results of clustering with constraints [6]. These results indicate that finding a *single* feasible clustering is intractable and it is not surprising that **counting** the number of clusterings/colorings is also intractable [11]. It is important to note that in constraint graphs, after performing a transitive closure over all ML constraints, each connected component can be represented as **one** node in Figure 1.

Chromatic Numbers. The chromatic number of a graph is the minimum number of colors required to color the vertices and its calculation is **NP**-Complete [11]. However the recent notion of fractional graph coloring, allows each vertex to be assigned multiple colors but still requires that a legal coloring has no two nodes connected by an edge sharing a common color. A graph has a legal b-fold coloring if color sets of size no more than b are assigned to each vertex and adjacent vertices have non-overlapping sets. Similarly, an *a:b*-coloring produces a b-fold coloring when a colors are available. Finally, the b-fold fractional chromatic number is the least value of a such that an *a:b*-coloring exists. We shall see that the fractional coloring problem is a relaxation of regular coloring and the fractional chromatic number can be calculated by a linear programming relaxation of an integer programming problem.

3. APPROACH 1: AN MCMC SAMPLER TO COUNT THE NUMBER OF FEASIBLE CLUSTERINGS

Though MCMC samplers have been used extensively in statistics, data mining and machine learning [9] the use of these algorithms **for counting** is novel having being relatively recently proposed in computer science theory [10]. Counting problems are a special application of samplers since the equilibrium distribution of the chain needs to be uniform which has the benefit of allowing strong performance bounds

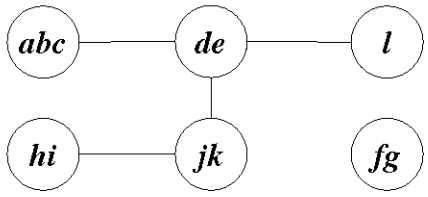

Figure 1: Constraint Graphs. A graphical representation of the feasibility problem for *ML(a,b), ML(a,c), ML(d,e), ML(f,g), ML(h,i), ML(j,k), CL(a,e), CL(i,j), CL(d,k), CL(e,l)*

to be created. In particular, provably efficient *rapidly mixing* samplers [10] are possible and there is no need to rely on typical statistical experimental approaches to test for convergence such as variance estimation or estimating the spectral radius. However, not all counting problems have an easily implementable uniform sampler as we shall see the counting of constrained clusterings does. Gomes et' al [7] note the difficulty in creating uniform samplers for SAT counting problems and that samplers not converging to a uniform equilibrium distribution can significantly over or under estimates counts.

The MCMC sampler we shall create at its core is a rejection sampler but unlike typical applications we shall use **many rejection samplers**, and many chains for each sampler and couple their results together. Our approach will be explained in four parts which are outlined in the following four subsections.

- Firstly, we shall create a **sufficient condition** when creating a feasible clustering is easy and use this to initialize our sampler. This sufficient condition ensures a completed connected state space.
- Next, we shall describe how to create **the rejection sampler** to determine the ratio $\frac{NF_{C-i}}{NF_{C-(i+1)}}$ which is the number of feasible clusterings for the entire constraint set less i specific CL constraints to the number of feasible clusterings for the same constraint set less **just one more CL constraint**. Note that the ratio will always be less then one since the denominator problem is a *relaxed* version of the numerator problem and the removal of a constraint can only increase the number of feasible clusterings. Note that ML constraints are not considered in our discussion because as mentioned earlier, they are merged into single nodes in the constraint graph (see Figure 1).
- Thirdly, we use a coupling (multiplying) approach to count the number of clusterings by multiplying the ratios of the rejection samplers together. It should be noted for clarity that we create many samplers and N_{chains} chains for each sampler and run each chain for N_{draws} draws after which it will reach equilibrium.
- Finally, we discuss how to determine values for N_{draws} (see equation 4) and N_{chains} (see equation 5) which can bound the error of our calculations.

We create a rejection sampler for all values of $i = 0 \ldots |C_{\neq}| - 1$ where $C_{\neq}$ is the number of CL constraints. For

the coupling approach to work deleting a constraint corresponds to deleting all equivalent constraints and effectively deletes an edge in the constraint graph. Consider two connected components formed by ML constraints: $\{a, b, c\}$ and $\{d, f, g\}$ and two CL constraints are $CL(a, d)$ and $CL(b, f)$ then the deletion of $CL(a, d)$ still leaves an edge between the two connected components in the constraint graph so $CL(b, f)$ would also need to be deleted to remove the edge. To simplify our notation, the constraint set C will contain only unique constraints, that is each CL constraint corresponds to only one unique edge in the constraint graph.

3.1 A Sufficient Condition

Consider the set of constraints in Figure 1. If the sufficient condition $k > \Delta$ holds where k is the number of clusters and Δ is the the most number of CL constraints on a single instance (i.e. the maximum degree of our constraint graph), then regardless of what order the instances are assigned to clusters, we can always produce a feasible clustering. **This is important since it guarantees a completely connected state space since in effect there is always one "free" cluster to move any point to.** This result is from the well known result, called Brooks's Theorem, in graph theory [11]. In our example (Figure 1) if $k = 4$ we can just proceed and assign the instances in any order and we will always find a feasible clustering. This sufficient condition does create a limitation when this sampler can be used but as discussed earlier typically a small number of constraints are used, hence Δ will be small so this should not be a limitation.

3.2 The MCMC Sampler

The sampler estimates the ratio of the number of feasible clusterings for a constraint set to the number of feasible clusterings for the same constraint set less one constraint. It takes in the original set of ML and CL constraints and removes one CL constraint (edge from the constraint graph). It then performs a random walk over the set of feasible clusterings for the **smaller** constraint set to reach equilibrium with the last state drawn being $\mathcal{X}$. The previously removed constraint is added back and if $\mathcal{X}$ satisfies this original larger constraint set then we add 1 to a success count otherwise we reject the sampled state. The proportion of successes gives us an estimate of the ratio. Figure 2 shows our algorithm, in a later section we shall show how to derive bounds for N_{draws} and N_{chains}. Note, this approach only works because the feasible clusterings of the larger constraint set is a proper subset of the feasible clusterings of the reduced constraint set. It can be viewed as a rejection sampler by noting that we are sampling from a larger set of constrained clusterings (given the smaller number of constraints) and then rejecting if a clustering does not also satisfy the additional constraint.

In our context the state space of the random walk is the space of all feasible clusterings and since each is equally important to us (we are just counting) the equilibrium distribution is the uniform distribution. An individual chain is essentially defined by step 8 in Figure 2 which chooses an instance at random and then assigns it to a new cluster. Clearly this satisfies the detailed balance requirement and will produce a finite, irreducible, aperiodic and hence ergodic chain.

Algorithm MCMC-Rejection Sampler

Input:
X: the set of points to cluster, k: number of clusters
$C_{\neq}$: the set of cannot-link constraints
$C_{=}$: the set of must-link constraints
N_{draws} and N_{chains}: as calculated from equations 4 and 5 respectively
Output: $\frac{NF_C}{(NF_{C-1})}$: The number of feasible clusterings ratio for a constraint set and the same set less one constraint.

1. $G(V, E)$=*GenerateConstraintGraph*$(X, C_{\neq}, C_{=}, k)$ (See section 2.)
2. Let Δ be the maximum degree of $G(V, E)$. If $k \leq \Delta$ exit *k not large enough.*
3. Create $C'_{\neq} = C_{\neq} - c(i, j)$ /*A single CL constraint is removed*/
4. $G'(V, E')$=*GenerateConstraintGraph*$(X, C'_{\neq}, C_{=}, k)$ (See section 2.)
5. $\mathcal{X}$=*GenerateInitialClustering*(X, G', k)
6. **for** c = 1 to N_{chains}
7. $Success_c = 0$
8. **for** d = 1 to N_{draws}
   (a) Let q be a random integer $\in [1, k]$
   (b) From $\mathcal{X}$ take a randomly chosen v and assign to a new cluster q to derive $\mathcal{X}_{Candidate}$
   (c) If Feasible($\mathcal{X}_{Candidate}$,$G'(V, E')$) then $\mathcal{X} = \mathcal{X}_{Candidate}$
   end
9. If Feasible($\mathcal{X}$,$G(V, E)$) then $Success_c = 1$ /*Note the random walk is on G' and we now test feasibility against G*/
10. **end**
11. **return** $\sum_i Success_i / N_{chains}$

Figure 2: A MCMC Rejection Sampler. The number of draws from each chain (N_{draws}) and the number of chains (N_{chains}) are specified in section 3.4.

3.3 Coupling Arrangement to Estimate Number of Feasible Clusterings

The above subsection describes how to create a single chain to calculate the ratio $\frac{NF_C}{NF_{C-1}}$, we now show how to use a series of such chains to calculate the total number of feasible clusterings NF_C. Our approach begins by creating a rejection sampler to calculate the ratio of feasible clusterings using all constraints to feasible clusterings for all constraints less one particular constraint. This is multiplied by the ratio of all feasible clusterings for the reduced constraint set to feasible clusterings for the same constraint set less one additional specific constraint and so on. We can then couple/multiply the results of the rejection samplers to achieve the final number of feasible clusterings by:

$$NF_C = \frac{NF_C}{NF_{C-1}} \times \frac{NF_{C-1}}{NF_{C-2}} \times \ldots \frac{NF_{C-(|C|-1)}}{NF_\emptyset} NF_\emptyset \quad (1)$$

Where NF_{C-i} is the number of feasible clusterings using the constraint set C less i specific CL constraints. The key to understanding why equation 1 is computable is to note that $NF_\emptyset = k!S(n,k)$ where n is the number of nodes in the constraint graph (i.e. CL constrained nodes) and that $S(n,k)$ can be calculated as described earlier in Section 2.

3.4 How Many Chains and How Long To Run Each Chain

Perhaps where MCMC algorithms for counting differ most than previous uses of samplers in learning is the existence of strong results for bounding the number of steps until convergence. This is possible since the sampler's equilibrium distribution is the uniform distribution. We use the measure of variation distance in equation 2 between the current distribution and the equilibrium distribution over the state space, Ω, starting from an **initial state** x to get an upper-bound on the convergence time (t) shown in equation 3.

$$\delta_x(t) = \frac{1}{2}\Sigma_{y\in\Omega}|P_x^t(y) - \pi(y)| \quad (2)$$

$$\tau_x(\epsilon) = \ min\{t : \delta_x(t') \leq \epsilon \ \forall t' \geq t\} \quad (3)$$

We use a result from [4] that the time for the chain to reach equilibrium is upper-bounded as shown in equation 4.

$$N_{draws} \ = \ \tau_x(\epsilon) \geq qlog(q\epsilon^{-1}) \quad (4)$$

Where q is the number of constrained instances (a large connected component counts as one point) and ϵ a measure of our choice. However, knowing how many draws to reach equilibrium is only half of our requirements. We must also know how many samples to draw after equilibrium is reached to bound the accuracy of our count. As is typical in the counting MCMC literature, we obtain our estimates from **multiple** Markov chains rather than sampling multiple times from the same chain. Skipping over much lengthy but standard math and substituting and rearranging a Chebyshev inequality we derive a bound shown in equation 5.

$$N_{chains} \geq \frac{1}{(\delta \times (\epsilon)^2)^{ln(ln(|C|))}} \quad (5)$$

where δ is our chance of failure.

This allows us to set the key parameters in our algorithm (Figure 2) to be a function of the error ($0 < \epsilon < 1$) of our estimate and chance of the bound failing ($0 < \delta < 0.5$). It is possible to show that this approach can be extended to a fully polynomial-time randomized approximation scheme and hence is capable of rapid mixing.

4. APPROACH 2: DETERMINING CLUSTER ASSIGNMENT DIFFICULTY USING FRACTIONAL CHROMATIC NUMBERS

Figure 1 shows the relationship between graph coloring and constrained clustering. With traditional graph coloring, each node is assigned exactly one color and the chromatic number of a graph is the minimum number of colors to color it. The chromatic number of a graph can then be determined by the following integer program. As before let the vertices of G (the constraint graph) be numbered from 1 to n and the maximal independent sets of vertices of G numbered from 1 to m. Recall an independent set is a subset of all vertices where no two vertices share an edge and hence in our context all vertices in an independent set can be assigned to the same cluster. Let A be a $n \times m$ matrix with $a_{i,j} = 1$ if the i^{th} vertex is in the j^{th} independent set, and 0 otherwise. Then the integer programming problem is:

$$\begin{aligned} &minimize \ \ \mathbf{x}.\mathbf{1} \\ subject\ to: \ \ &A\mathbf{x} \geq \mathbf{1}, \mathbf{x} \geq \mathbf{0} \end{aligned} \quad (6)$$

The indicator vector $\mathbf{x}$ represents which independent sets are required and the objective function just counts the number required. The first constraint requires each vertex to be part of at least one independent set (cluster) and the second to prevent the trivial solution. We can then view fractional graph coloring as a relaxation of this problem where $\mathbf{x} \in [0,1]$ and hence a vertex can be assigned fractionally to different independent sets (therefore having multiple colors) and we can easily see this can be solved as a linear programming problem. As we shall see the number of fractional colors associated with an instance is a measure of how **easy** it is to assign this point and the b-fold chromatic number a measure of how difficult the entire constraint set is to satisfy.

5. EXPERIMENTAL INSIGHTS

We use four core algorithms in our experiments. They are chosen because they are some of the seminal work in the field and hence are widely used with stable freely available implementations. Furthermore, they cover a wide range of different ways of using constraints. The **CKM** algorithm is the original COP-k-means algorithm [13] that attempts to satisfy **all** constraints. The **PKM** algorithm of Basu and collaborators [1] allows constraints to be violated but tries to minimize this number. The related **MKM** algorithm is a metric learning algorithm that was shown to perform comparable or better than the seminal work in the field [15]. After the metric space is learnt, we apply regular unconstrained k-means. Finally the **MPKM** algorithm combines metric learning and constraint satisfaction by modeling the constraints as enforcing a prior using a hidden Markov random field. Similarly, we choose our four data sets due to their variety in size, attribute type and most importantly number of extrinsic classes. We always set k to equal the number of extrinsic labels (unless otherwise specified) and generate constraint randomly by selecting pairs of instances at random and comparing them: if the labels agree then a

Table 1: Fraction of 1000 randomly created 25-constraint sets that caused a drop in accuracy (measured by the Rand index), compared to an unconstrained run with the same centroid initialization.

	Algorithm			
Data	**CKM**	**PKM**	**MKM**	**MPKM**
Glass	31%	7%	16%	3%
Ionosphere	26%	68%	3%	80%
Iris	27%	23%	31%	41%
Wine	35%	38%	82%	69%

must-link is generated otherwise a cannot-link is generated. We use the following data sets with number of extrinsic labels (k), instances (n) and dimensions (m) in parentheses, Iris($k = 3$, $n = 150$, $m = 4$), Wine($k = 3$, $n = 178$, $m = 13$), Ionsphere($k = 2$, $n = 351$, $m = 34$) and Glass($k = 6$, $n = 214$, $m = 10$).

In this section we investigate several questions with our proposed work but no doubt there are other important questions. We begin by exploring a counter-intuitive result published earlier [5] that showed that constraints can **hurt** performance of algorithms. In that work, they showed that even though on average (over many constraint sets of the same size) the accuracy of a variety of algorithms (as measured by the Rand index between the ground truth partition defined by the labels and the partition/clustering found by the algorithms) increased with the amount of constraints given to the algorithm, there were a significant number of individual constraint sets where performance was degraded (but not enough to bring down the average significantly). This leads to our first experimental question.

Q1: For those constraint sets where performance is ***worse*** *than not using any constraints, was there a significant difference in the number of feasible clusterings compared to constraint sets that helped clustering performance.*

We verified earlier results [5] and see in Table 1 that the fraction of these 1000 trials that caused **a decrease in clustering accuracy** (compared to using no constraints) is significant. Since each trial involved the same initialization mechanism for both unconstrained and constraint experiments, performance differences are due to the use of constraints.

A valid question is then: *Do the constraint sets where the algorithms perform poorly, correspond to those with fewer feasible clusterings?* To answer this question we split the 1000 constraint sets for each algorithm into those where the the constraint set produced no change or a decrease in accuracy (hurt) and those that produced an increase in accuracy compared to not using constraints (help). We then used our sampler to approximately count how many feasible clustering (on average) each sub-population contained. Results in a comparative bar chart are in Figure 3 and show that constraint sets that hurt clustering accuracy have a smaller number of feasible clusterings on average than those that helped.

Q2: Is the b*-fold Chromatic Number A Useful Measure of Constraint Difficulty?*

In these experiments, we attempt to see how the Rand index varies depending on the minimum b-fold chromatic number. For each of our data sets we generate 100 constraint sets of size 25. For each constraint set, we solved our linear program to determine the fractional chromatic number. We

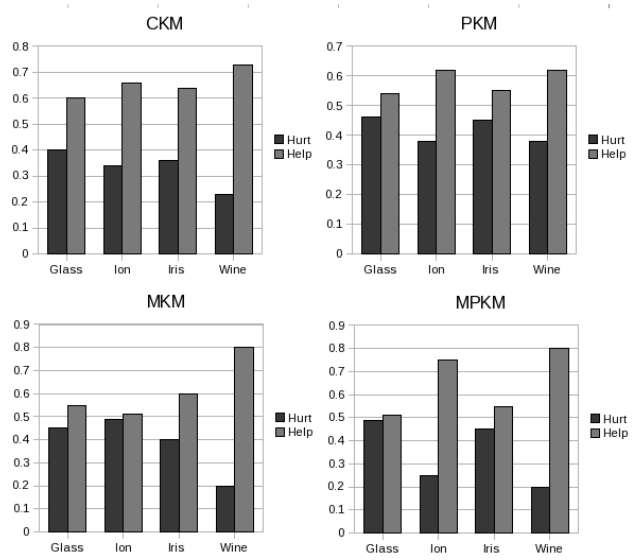

Figure 3: The average ratio of the number of feasible clusterings to total clusterings for two sub-populations: constraint sets that help performance and those that hurt performance.

plot this as a scatter plot with the x-axis being the inverse of fractional chromatic number (the larger the easier the graph is to color) and the y-axis the Rand index. The scatter plot for each data set is shown in Figure 4 and shows a strong positive correlation between the x and y axes.

6. SEVERAL USES OF OUR WORK

Our two approaches allow us to estimate how difficult a constraint set is to satisfy which has many potential applications. In this section we outline several uses of our algorithms. The first is useful when the user has the possibility to generate many different constraint sets easily and allows them to generate the most useful set of constraints (see Figure 5). The second is useful when the user is given just a single constraint set and can remove constraints to improve the algorithm performance (see Figure 6).

6.1 Use 1: Constraint Generating Methods

Our experimental insights regarding the ability of the algorithm to recover the set partition the constraints were generated from can be summarized as follows. Constraint sets that allow a too large **infeasible** set of clusterings cause the CKM, PKM, MKM and MPKM algorithms to perform poorly with respect to the Rand index (see Figure 3). However, as others have shown the more constraints given to the algorithm the better. **Therefore, our aim is to generate as many constraints as possible without overly restricting the algorithm.**

To achieve this we create a **balanced** sampling approach for *CL* constraints as follows. Before we generate the i^{th} *CL* constraint involving instance j, all other instances must have $i-1$ *CL* constraints on them. In this way we make the degree of nodes across the constraint graph uniform. *ML* constraint generation does not change. We then compare this approach of generating constraints to the typical ap-

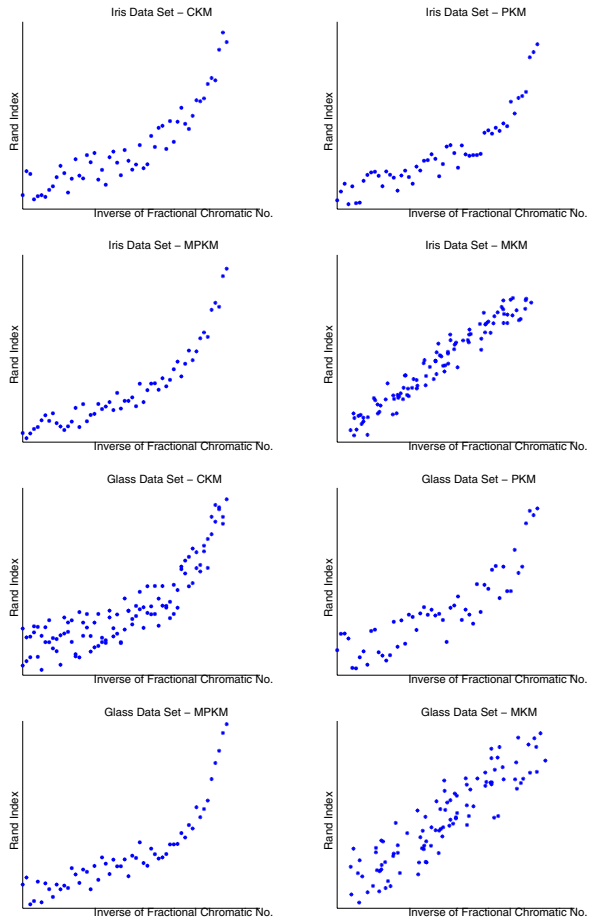

Figure 4: The fractional chromatic number inverse(x-axis) (the larger the easier the problem) versus the Rand index (y-axis) for one hundred constraint sets for a variety of algorithms. Similar relationships hold for other data sets.

proach of randomly selecting points on which to generate constraints. We ensure generating equal number of *ML* and *CL* constraints for both styles of constraint generation.

We plot the average Rand index (over 100 constraint sets) for increasing amounts of constraints for a variety of algorithms and data sets for the two methods. We found that on average the new approach of generating constraints outperformed the traditional approach by a statistically significant amount at the 99% confidence interval using a student t-test after just twenty constraints. Results are shown in Figure 5. We see that initially the random approach of generating constraints performs similar to our own since for small number of constraints it is unlikely that any balancing is needed. However, when constraint sets became larger our balanced sampling approach produces significantly better results. We tested our work on two differing styles of constrained clustering algorithm: **CKM** which attempts to satisfy all constraints and **MPKM** which both learns a distance metric and satisfies as many constraints as possible.

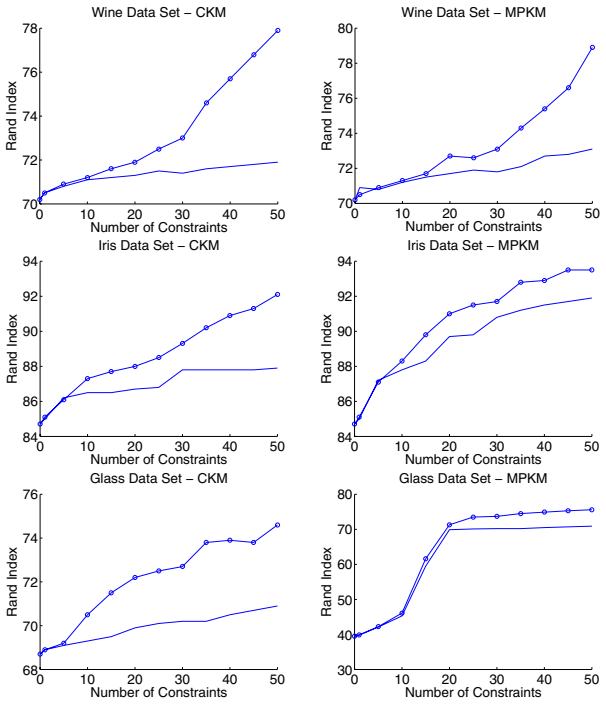

Figure 5: The Rand index versus the number of constraints. The solid line (—) is for constraints generated by random selection while the other line (-o-) is our balanced generation approach. The Rand index is averaged over 100 constraint sets.

6.2 Use 2: Pruning Constraint Sets

An output of the linear program solution of equation 6 is effectively a measure of how flexible (or conversely how difficult) a point is to assign. If the solution indicates that point i belongs to many independent sets and point j only one, then it is more difficult to assign the latter than the former since there are fewer options. To the graph of the constrained points (recall edges indicate cannot link constraints) we add their k furthest neighbors and corresponding edges (indicating unlikely points to be grouped with). We then solve equation 6 to get a better estimate of the difficulty of assigning each of the constrained points. In our experiments we use the five furthest neighbors.

A natural method of pruning constraint sets is then to remove the points that are part of the least number of independent sets and corresponding constraints they are part of. This set of points are those though potentially useful, are difficult for algorithms to handle since there is little freedom in assigning them to clusters. This is particularly important since most constrained clustering algorithms are iterative and repetitively attempt to assign a point to a cluster and incrementally build up the clustering. Our experimental results in Figure 6 show the Rand index increases as the constraints are pruned. A valid question for future work is when to stop pruning which we side-step by generating many constraints and only pruning a few.

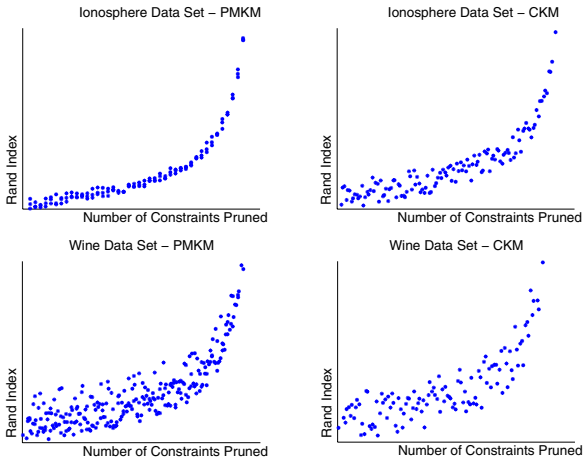

Figure 6: Typical results of the Rand index versus the number of constraints pruned (from 1 through 10) from a constraint set of size 100. The results over 100 experiments are shown.

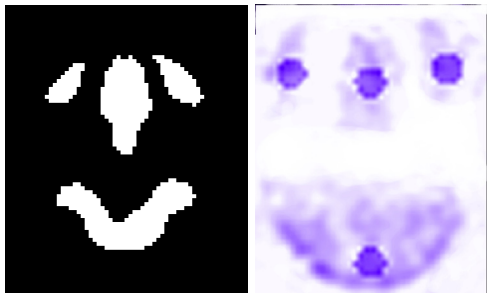

Figure 7: The idealized default mode network (left) from which many constraints can be generated. The resultant network found (right) with constraints actually used (i.e. not pruned) shown in darker color.

6.3 Use 3 - Simplest Geometric Constraints

Functional Magnetic Resonance Imaging (fMRI) is extensively used for capturing brain activity as it processes information in real time. A key science question is clustering the voxels/pixels in the images (represented as time sequences) into two groups (correlated or networked versus other or background). Constraints can help guide clustering algorithms since they can be used to represent geometric properties [13][6] such as maximum diameter and maximum separation. Consider the default mode network (DMN) shown in Figure 7 (left). Each of the four regions can be represented as a large number of must-link constraints and we can require that the surrounding areas not merge into the regions using cannot-link constraints. However, this will generate a huge number of constraints that can easily over-constrain the algorithms. In all our experiments with this data using all generated constraints the CKM algorithm which attempts to satisfy all constraints did not converge (being over-constrained) and the metric learning algorithms (such as MKM) returned poor results.

We apply our constraint pruning approach described in the previous sub-section to leave just 10% of all possible constraints and the resultant clustering (and subset of constraints produced) are shown in Figure 7 (right). We see two clusters (light blue: DMN, white: background) are discovered and a reasonable subset set of constraints are produced (shown in dark blue).

7. CONCLUSION AND FUTURE WORK

In this work we proposed two answers to the problem:" Which constraint sets help constrained clustering algorithms ?". We posed the novel question: "Can the number of feasible clusterings that satisfy all constraints be counted?" Though this problem is intractable in general, we showed that under a reasonable sufficient condition a simple rapidly mixing MCMC sampler can approximately count the number of feasible clusterings. There are many uses of this particular sampler for analysis and we focused on the open question: "Why do some constraint sets adversely effect clustering performance to the point that it would have been better **not** to have used constraints?" We showed that those constraint sets that hurt performance had far less feasible clusterings than those that helped performance. We then proposed a method of creating larger constraint sets without over-constraining the algorithms and showed it was superior than the traditional approach (see Figure 3).

We also showed that the difficultly in fractionally coloring the constraint graph was reflective of how useful the constraint set was to the clustering algorithm's performance with respect to accuracy (see Figure 4). To do this we created a LP that determine the minimum b-fold chromatic number. A side benefit of the LP solution is that it tells us how many independent sets a constrained point belonged to. The fewer the number the less available clusters the algorithm has to place them in without violating constraints. We showed that removing/pruning constraints on difficult to assign points produced improved performance on the algorithms we studied and that for real world fMRI data the approach is a reasonable way of pruning the many thousands of constraints that can be generated from geometric properties to enforce on the clustering (see Figure 7).

We can therefore conclude that all constraint sets are not generated equally, some more restrict the possible clusterings and understanding which do is important for algorithm designers and practitioners alike. We hope that this work will make some progress towards this and other important questions.

8. ACKNOWLEDGMENTS

The authors gratefully acknowledge support of this research from the NSF (IIS-0801528) and ONR (N000140910712).

9. REFERENCES

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

[2] *Constrained Clustering: Advances in Algorithms, Theory, and Applications* Editors: Sugato Basu, Ian Davidson, Kiri L. Wagstaff CRC Press

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

[4] R.Bubley and M. Dyer., Path coupling: A technique for proving rapid mixing in Markov chains, Proceedings of 38^{th} FOCS, 1997.

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

[6] 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.

[7] C. Gomes, A. Sabharwal, and B. Selman, From Sampling to Model Counting, IJCAI, Hyderabad, India, Jan 2007.

[8] L. Goodman, The Variance of the Product of K Random Variables, JASA V 57, No 297 p 54-60, 1962.

[9] R. Neal, Probabilistic Inference Using Markov Chain Monte Carlo Methods, Technical Report CRG-TR-93-1, Department of Computer Science, University of Toronto.

[10] M. Jerrum, A. Sinclair, "The MCMC method: an approach to approximate counting and integration", In *Approximations for NP-hard Problems*, 1996.

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

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

[13] K. Wagstaff, C. Cardie, S. Rogers & S. Schroedl, "Constrained k-means Clustering with Background Knowledge", ICML, 2001.

[14] K. Wagstaff, *Intelligent Clustering with Instance-Level Constraints*, Ph.D. Dissertation, Cornell University, 2002.

[15] E. Xing, A. Ng, M. Jordan, and S. Russell. Distance metric learning, with application to clustering with side-information. NIPS 15, 2003.