Graph-Based Clustering with Constraints

Rajul Anand and Chandan K. Reddy

Department of Computer Science,
Wayne State University, Detroit, MI, USA
rajulanand@wayne.edu, reddy@cs.wayne.edu

Abstract. A common way to add background knowledge to the clustering algorithms is by adding constraints. Though there had been some algorithms that incorporate constraints into the clustering process, not much focus was given to the topic of graph-based clustering with constraints. In this paper, we propose a constrained graph-based clustering method and argue that adding constraints in distance function before graph partitioning will lead to better results. We also specify a novel approach for adding constraints by introducing the distance limit criteria. We will also examine how our new distance limit approach performs in comparison to earlier approaches of using fixed distance measure for constraints. The proposed approach and its variants are evaluated on UCI datasets and compared with the other constrained-clustering algorithms which embed constraints in a similar fashion.

Keywords: Clustering, constrained clustering, graph-based clustering.

1 Introduction

One of the primary forms of adding background knowledge for clustering the data is to provide constraints during the clustering process [1]. Recently, data clustering using constraints has received a lot of attention. Several works in the literature have demonstrated improved results by incorporating external knowledge into clustering in different applications such as document clustering, text classification. The addition of some background knowledge can sometimes significantly improve the quality of the final results obtained. The final clusters that do not obey the initial constraints are often inadequate for the end-user. Hence, adding constraints and respecting these constraints during the clustering process plays a vital role in obtaining desired results in many practical domains. Several methods are proposed in the literature for adding instance-level and cluster-level constraints. Constrained versions of partitional [19,1,7], hierarchical [5,13] and more recently, density-based [17,15] clustering algorithms have been studied thoroughly. However, there has been little work in utilizing the constraints in the graph-based clustering methods [14].

1.1 Our Contributions

We propose an algorithm to systematically add instance-level constraints to the graph-based clustering algorithm. In this work, we primarily focused our attention to one such popular algorithm, CHAMELEON, an overview of which is provided in section 3.2. Our contributions can be outlined as follows:

J.Z. Huang, L. Cao, and J. Srivastava (Eds.): PAKDD 2011, Part II, LNAI 6635, pp. 51–62, 2011.

- Investigate the appropriate way of embedding constraints into the graph-based clustering algorithm for obtaining better results.
- Propose a novel distance limit criteria for must-links and cannot-links while embedding constraints.
- Study the effects of adding different types of constraints to graph-based clustering.

The remainder of the paper is organized as follows: we briefly review the current approaches for using constraints in different methods in Section 2. In Section 3, we will describe the various notations used throughout our paper and also give an overview of a graph-based clustering method, namely, CHAMELEON. Next, we propose our algorithm and discuss our approach regarding how and where to embed constraints in Section 4. We present several empirical results on different UCI datasets and comparisons to the state-of-the-art methods in Section 5. Finally, Section 6 concludes our discussion.

2 Relevant Background

Constraint-based clustering has received a lot of attention in the data mining community in the recent years [3]. In particular, instance-based constraints have been successfully used to guide the mining process. Instance-based constraints enforce constraints on data points as opposed to ϵ and δ constraints which work on the complete clusters. The ϵ-constraint says that for cluster X having more than two points, for each point $x \in X$, there must be another point $y \in X$ such that the distance betweeen x and y is at most ϵ. The δ-constraint requires distance between any two points in different clusters to be at least δ . This methodology has also been termed as semi-supervised clustering [9] when the cluster memberships are available for some data. As pointed out in the literature [19,5], even adding a small number of constraints can help in improving the quality of results.

Embedding instance-level constraints into the clustering method can be done in several ways. A popular method of incorporating constraints is to compute a new distance metric and perform clustering. Other methods directly embed constraints into optimization criteria of the clustering algorithm [19,1,5,17]. Hybrid methods of combining these two basic approaches are also studied in the literature [2,10]. Adding instance-level constraints to the density-based clustering methods had recently received some attention as well as [17,15]. Inspite of the popularity of graph-based clustering methods, not much attention is given to the problem of adding constraints to these methods.

3 Preliminaries

Let us consider a dataset D, whose cardinality is represented as $\|\mathrm{D}\|$. The total number of classes in the dataset are K . Proximity graph is constructed from this dataset by computing the pair-wise Euclidean distance between the instances. A user-defined parameter k is used to define the number of neigbors considered for each data point. The hyper-graph partitioning algorithm generates intermediate subgraphs (or sub-clusters) to be formed which are represented by κ.

3.1 Definitions

Given a dataset D with each point denoted as (x, y) where x represents the point and y represent the corresponding label, we define constraints as follows:

Definition 1: ***Must-Link Constraints(ML)***: Two instances (x_1, y_1) and (x_2, y_2) are said to be must-link constraints, if and only if, $y_1 = y_2$ where $y_1, y_2 \in K$.

Definition 2: ***Cannot-Link Constraints(CL)***: Two instances (x_1, y_1) and (x_2, y_2) are said to be cannot-link constraints, if and only if, $y_1 \neq y_2$, where $y_1, y_2 \in K$.

Definition 3: ***Transitivity of ML-constraints***: Let X, Y be two components formed using ML-constraints. Then, a new ML-constraint $(x_1 \overset{must-link}{\rightarrow} x_2)$ where $x_1 \in X$ and $x_2 \in Y$ introduces the following new constraints: $(x_i \overset{must-link}{\rightarrow} x_j)\ \forall x_i, x_j$ where $x_i \in X$ and $x_j \in Y$, $i \neq j$,$X \neq Y$.

Definition 4: ***Entailment of CL-constraints***: Let X, Y be two components formed using ML-constraints. Then, a new CL-constraint $(x_1 \overset{cannot-link}{\rightarrow} x_2)$, where $x_1 \in X$ and $x_2 \in Y$ introduces the following new CL-constraints: $(x_i \overset{cannot-link}{\rightarrow} x_j)\ \forall x_i, x_j$ where $x_i \in X$ and $x_j \in Y$, $i \neq j$,$X \neq Y$.

3.2 Graph-Based Hierarchical Clustering

We chose to demonstrate the performance of adding constraints to the popularly studied and practically successful CHAMELEON clustering algorithm. Karypis et al. [11] proposed CHAMELEON algorithm which can find arbitrarily shaped, varying density clusters. It operates on sparse graphs containing similarity or dissimilarity between data points. Compared to various graph-based clustering methods [18] such as Minimum Spanning Tree clustering, OPOSSUM, ROCK, SLINK, *CHAMELEON is superior because it incorporates the best features of graph-based clustering* (like similarity measure on vertices as in ROCK) and the hierarchical clustering part which is similar or better than SLINK. These features make CHAMELEON attractive to add constraints to obtain better results. Moreover, *CHAMELEON outperforms other algorithms like CURE* [11] and also *density-based methods like DBSCAN* [18]. Thus, we believe adding constraints to CHAMELEON will not only give better results but also provide some insights on the performance of other similar algorithms in the presence of constraints.

Unlike other algorithms which use a given static modeling parameters to find clusters, CHAMELEON find clusters by dynamic modeling. CHAMELEON uses both closeness and interconnectivity while identifying the most similar pair of clusters to be merged. CHAMELEON works in two phases. In the first phase, it finds the *k*-nearest neighbors based on the similarity between the data points. Then, using an efficient multi-level graph partitioning algorithm (such as METIS [12]), sub-clusters are created in such a way that similar data points are merged together. In the second phase, these sub-clusters are combined by using a novel agglomerative hierarchical algorithm. Clusters are merged using RI and RC metrics defined below. Let X, Y be two clusters. Mathematically, Relative Interconnectivity (RI) is defined as follows:

$$RI = \frac{EC(X,Y)}{\frac{1}{2}(EC(X) + EC(Y))} \tag{1}$$

where $EC(X,Y)$ is the sum of edges that connects clusters X and Y in the *k*-nearest neighbor graph. $EC(X)$ is the minimum sum of the cut-edges if we bisect cluster X; and $EC(Y)$ is the minimum sum of the cut-edges if we bisect cluster Y. Let l_x and l_y represents size of the clusters X and Y respectively. Mathematically, Relative Closeness (RC) is defined as follows:

$$RC = \frac{\overline{S}_{EC}(X,Y)}{\frac{l_x}{l_x+l_y}\overline{S}_{EC}(X) + \frac{l_y}{l_x+l_y}\overline{S}_{EC}(Y)} \tag{2}$$

where $\overline{S}_{EC}(X,Y)$ is the average weight of edges connecting clusters X and Y in *k*-nearest neighbor graph. $\overline{S}_{EC}(X)$, $\overline{S}_{EC}(Y)$ represents the average weight of edges if the clusters X and Y are bisected correspondingly.

There are many schemes to account for both of the measures. The function used to combine them is $(RI \times RC^{\alpha})$. Here another parameter α is included so as to give preference to one of the two measures. Thus, we maximize the function:

$$argmax_{\alpha \in (0,\infty)}(RI \times RC^{\alpha}) \tag{3}$$

4 Constrained Graph-Based Clustering

CHAMELEON, like other graph-based algorithms, is sensitive to the parameters as a slight change in similarity values can both dramatically increase or decrease the quality of the final outcome. For CHAMELEON, changes in similarity measures might result in different k-nearest neighbors. Overlapping clusters or clusters with very minimal inter-cluster distance sometimes leads to different class members in the same cluster. In this work, the primary emphasis is to demonstrate that adding constraints to graph-based clustering can potentially avoid this problem at least sometimes, if not always. Our basis for this assumption, is the transitivity of ML constraints and the entailing property of CL constraints (Section 3.1).

4.1 Embedding Constraints

Using distance (or dissimilarity) metric to enforce constraints [13] was claimed to be effective in practice, despite having some drawbacks. The main problem is caused due to setting the distance to zero between all the must-linked pair of constraints. i.e., Let (p_i, p_j) be two instances in a must-link constraint then,

$$distance(p_i, p_j) = 0$$

To compensate for distorted metric, we run all-pairs-shortest-path algorithm so that new distance measures is similar to the original space. If we bring any two points much closer to each other, i.e.

$$\lim_{n \to distance(p_i,p_j)} distance(p_i, p_j) - n = \eta \tag{4}$$

At the first look, it may seem that this minute change will not affect the results significantly. However, after running all-pairs-shortest-path algorithm, the updated distance matrix in this case, will respect the original distance measures better than setting the distance to zero. Similarly for cannot-link constraints, let (q_i, q_j) be a pair of cannot-link constraints, then the points q_i and q_j are taken apart as far as possible. i.e.,

$$\lim_{n \to \infty} distance(q_i, q_j) + n = \lambda \tag{5}$$

Thus, by varying the values of η and λ, we can push and pull away points reasonably. It seems that this might create a problem for finding optimal values of η and λ. However, our preliminary experiments show that the basic limiting values for these parameters is enough in most of the cases. This addition of constraints (and thus the manipulation of the distance matrix) can be performed in the CHAMELEON algorithm primarily in any of the two phases. We can add these constraints before (or after) the graph partitioning step. After the graph partitioning, we can add constraints during agglomerative clustering. However, *we prefer to add constraints before graph partitioning* primarily due to the following reasons:

- When the data points are already in sub-clusters, enforcing constraints through distance will not be beneficial unless we ensure that all such constraints are satisfied during the agglomerative clustering. However, constraint satisfaction might not lead to convergence every time. Especially with CL constraints, even determining whether satisfying assignments exist is NP-complete.
- Intermediate clusters formed are on the basis of original distance metric. Hence, RI and RC on the original distance metric will get undermined by the new distance updation through constraints.

4.2 The Proposed Algorithm

Our approach for embedding constraints into the clustering algorithm is through learning a new distance (or dissimilarity) function. This measure is also adjusted to ensure that the new distance (or dissimilarity) function respects the original distance values to a maximum extent for unlabeled data points. We used Euclidean distance for calculating dissimilarity. For embedding constraints, an important and a intuitive question is: where to embed these constraints to achieve the best possible results? As outlined in the previous section, we choose to embed constraints in first phase of CHAMELEON. Now, we will present a step-by-step discussion of our algorithm.

Using Constraints. Our algorithm begins by using constraints to modify the distance matrix. To utilize properties of the constraints (Section 3.1) and to restore metricity of the distances, we propogate constraints. The must-links are propogated is done by running the fast version of all-pairs-shortest-path algorithm. If u, v represents the source and destination respectively, then the shortest path between u, v involves only points u, v and x, where x must belong to any pair of ML constraints. Using this modification, the algorithm runs in $O(n^2 m)$ (here m is the number of unique points in ML). The complete-link clustering inherently propagates the cannot-link constraints. Thus, there is no need to propagate CL constraints during Step 1.

Algorithm 1. Constrained CHAMELEON(CC)

Input: Dataset D, Set of must-link constraints $ML = \{ml_1, ..., ml_n\}$, Set of cannot-link constraints $CL = \{cl_1, ..., cl_n\}$, Number of desired clusters *K*, Number of nearest neighbors *k*, Number of intermediate clusters κ, Significance factor for RI or RC α.

Output: Set of *K* clusters

Step 1: **Embed Constraints**
for all (p1,p2) $\in$ ML **do**
    $\lim_{n \to distance(p_1,p_2)}$ distance (p_1, p_2) - n = η
end for
fastAllPairShortestPaths(DistanceMatrix)
for all (q1,q2) $\in$ CL **do**
    $\lim_{n \to \infty}$ distance (p_1, p_2) + n = λ
end for
Step 2: k-nn = **Build k-nearest neighbor graph**
Step 3: **Partition the k-nn graph into κ clusters using edge cut minimization**
Step 4: **Merge the κ clusters until K number of clusters remain by maximizing $RI \times RC^{\alpha}$ as criteria for merging clusters**

Importance of Constraint Positioning. Imposition of CL constraints just before Stage 4 rather than in Stage 1 might seem reasonable. We used CL constraints in Stage 1 due to our experimental observations stated below:

1. Hyper-graph partitioning with constraints is often better than constrained agglomerative clustering, when we are not trying to satisfy constraints in either one of them.
2. Clusters induced by graph partitioning have stronger impact on the final clustering solution.

After Step 1, we create the *k* nearest neighbor graph (Step 2) and partition the *k*-nn using a graph partitioning algorithm (METIS). The κ number of clusters are then merged using the agglomerative clustering where the aim is to maximize the product $(RI \times RC^{\alpha})$. Complete-link agglomerative clustering is used to propogate CL constraints embedded earlier. The cut-off point in dendrogram of the clustering is decided by parameter K (See Algorithm 1). The time complexity of unconstrained version of our algorithm is $O(n\kappa + nlogn + \kappa^2 log\kappa)$ [11]. The time complexity of Stage 1 consists of adding constraints which is O(l) (l = $\|ML\| + \|CL\|$) and $O(n^2 m)$ for propagation of ML constraints. Thus, overall complexity of $O(n^2 m)$ for Stage 1. Therefore, time complexity of our algorithm is finally $O(n\kappa + nlogn + n^2 m + \kappa^2 log\kappa)$.

5 Experimental Results

We will now present our experimental results obtained using the proposed method on benchmark datasets from UCI machine Learning Repository [8]. Our results on various versions of Constrained CHAMELEON(CC) were obtained with same parameter settings for each dataset. These parameters were not tuned particularly for CHAMELEON, however we did follow some specific guidelines for each dataset to obtain these parameters. We used the same default settings for all the internal parameters of the METIS

hyper-graph partitioning package except κ, which is dataset dependent. *We did not compare our results directly with constrained hierarchical clustering, since CC itself contains hierarchical clustering, which will be similar or better than stand-alone hierarchical clustering algorithms.* Instead, we compared with those algorithms which embed constraints into the distance function in the same manner as our approach. Our CC with fixed values of (0,∞) for (η, λ) is similar to [13] except that we have graph-partitioning step on nearest-neighbor graph before agglomerative clustering. So, we ruled out this algorithm and instead compared our results with MPCK-means [4] as this algorithm also embeds constraints in the distance function. MPCK-means incorporates learning of the distance function on labeled data and on the data affected by constraints in each iteration. Thus, it learns different distance function for each cluster.

For the performance measure, we used the Rand Index Statistic [16], which measures the agreement between two sets of clusters X and Y for the same set of n objects as follows: $R = \frac{a+b}{\frac{n}{2}}$, where a is the number of pairs of objects assigned to the same cluster in both X and Y , and b is the number of pairs of objects assigned to different clusters in both X and Y. *All the parameter selection is done systematically*. For all the clustering results, *K* is set to be the true number of classes in the dataset. The value of α is chosen between 1-2 with the increments of 0.1. We ran some basic experiments on CHAMELEON for each dataset, to figure out the effect of α on the results. We choose the particular value of α for each dataset which can potentially produce better result. We used similar procedure for *k* and κ. It is important to note that κ is dataset dependent parameter among all the other parameters. We are assuming that at least 10 data points should be present in a single cluster. Thus $K \leq \kappa \leq \|D\|/10$. We used the class labels of the data points to generate constraints. We randomly select a pair of data points and check their labels: if the labels are same, they are denoted as must-link, else they are denoted as cannot-link. To assess the impact of the constraints on the quality of the results, we varied the number of constraints. We generated results for ML only, CL only and ML, CL constraints. The complete dataset is used to randomly select data points for constraints, thus removing any bias towards the generated constraints.

Table 1. Average Rand Index values for 100 ML + CL constraints on UCI datasets

Datasets	**Instances**	**Attributes**	**Classes**	**MPCK-means**	**CC(p=1)**	**CC(fixed)**
Ionosphere	351	34	2	0.5122	0.5245	**0.5355**
Iris	150	4	3	0.6739	0.7403	**0.7419**
Liver	345	6	2	0.5013	0.5034	**0.5097**
Sonar	208	60	2	0.5166	0.5195	**0.5213**
Wine	178	13	3	0.6665	0.6611	**0.7162**
Average				0.5741	0.5898	**0.6049**

We used five UCI datasets in our experiments as shown in Table 1. Average Rand Index values for 100 Must-link and Cannot-link constraints clearly outlines that on most occasions, MPCK-means [4] is outperformed by both the variants of CC. CC(fixed) performed marginally better than CC(p=1). Also, we only show results for CC(p=5) and CC(p=15), since the results of CC(p=1) and CC(p=10) are similar to the other two.

For each dataset, we randomly select constraints and run algorithm once per constraint set. This activity is done 10 times and we report the average Rand-Index value for all the 10 runs. We used this experimentation for all the variants of CC and MPCK-means. The results are depicted in Figs. 1-3. We state that the distance value for must-links and cannot-links can be varied instead of fixed values like 0 and ∞ correspondingly. The CC(fixed) uses (0,∞) for distance measures. Ideally, the values of (η, λ) could be anything close to extreme values of 0 and ∞, yet they have to be quantifiable. In order to quantify them in our experiments, we defined as follows:

$$\lambda = D_{max} * 10^{p} \tag{6}$$

$$\eta = D_{min} * 10^{-p} \tag{7}$$

where D_{max}, D_{min} represents maximum and minimum distance values in the data matrix respectively. In order to study the effect of p, we varied it's values: p = 1, 5, 10 and 15. Thus, we have CC(p = 1), CC(p = 5), CC(p = 10) and CC(p = 15) referring to different values of (η, λ). It is interesting to note that, *for different values of p, distance values for constraints are different for each dataset* due to different minimum and maximum distance values. In this manner, *we respect the metricity of original distances and vary our constraint values accordingly*.

We tried various parameter settings and found only few selected ones to be making some significant difference in the quality of the final results. It is also important to note that these settings were found by running the basic CHAMELEON algorithm rather than CC. This is because, finding optimal parameters for CC using various constraints will be constraints-specific and it will not truly represent the algorithmic aspect. We then run CC using a few selected settings for all the variants of CC using all constraints size and finally report the average values specific to one set of parameters only showing better performance on average across all CC variants. The individual settings of parameters (k, κ, α) for each dataset shown in results are as follows: Ion(19,10,1.2), Iris(9,3,1), Liver(10,5,2), Sonar(6,3,1) and Wine(16,3,1). *In summary, we selected the best results obtained by the basic version of the CHAMELEON algorithm, and have shown that these best results can be improved by adding constraints*.

We observed across all the variants of CC and MPCK-means for all datasets consistently that the performance decreases as the number of constraints increase, except in some prominent cases (Figs. 1(d),2(a),2(b) and 3(d)). This observation is consistent with the results outlined in the recent literature [6]. We stated earlier that we did not attempt to satisfy constraints implicitly or explicitly. However, we observed that during Step 3 of Algorithm 1, for fewer constraints, most of the times the constraint violation is zero in the intermediate clusters, which is often reflected in the final partitions. As the number of constraints increase, the number of constraint violations also increase. However, on an average, violations are roughly between 10%-15% for must-link constraints, 20%-25% for cannot-link constraints, and about 15%-20% for must-links and cannot-links combined. We also observed that few times, the constraint violations are reduced after Step 4, i.e., after the final agglomerative clustering. Thus, we can conclude that the effect of constraints is significant in Step 3 and we re-iterate that embedding constraints earlier is always better for CC.

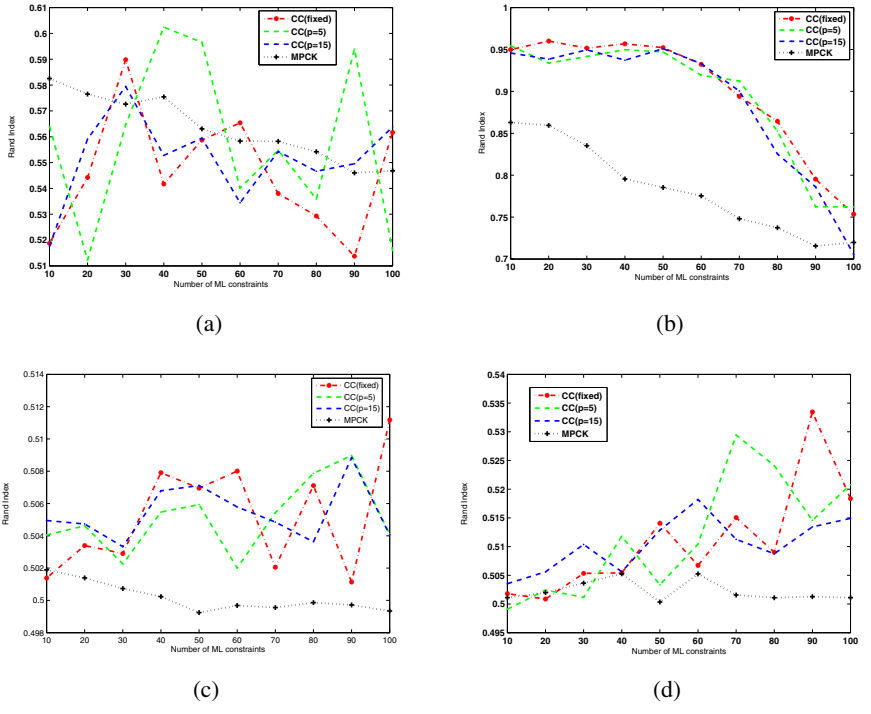

Fig. 1. Different versions of Constrained CHAMELEON(CC) compared with MPCK-means using Rand Index values averaged over 10 runs with ML constraints on different UCI datasets (a) Ionosphere (b) Iris (c) Liver and (d) Sonar

Overall, different variants of our algorithm CC outperformed MPCK-means. Iris and liver datasets are examples where for all combinations of constraints, CC results are clearly much better than MPCK-means. In the rest of the two datasets, CC performed nearly similar to MPCK-means. Only in some cases, MPCK-means performed slightly better than CC variants as shown in Figs. 1(a), 2(a) and 2(d). Even in these particular scenarios, at least one variant of CC outperformed (or nearly matched) the result of MPCK-means. Surprisingly, CC(fixed) was slightly better or worse compared to the other variants of CC. A direct comparison of CC(fixed) with MPCK-means reveals that only in two cases (Figs. 2(a) and 2(d)), it outperformed CC(fixed). In the rest of the scenarios, CC(fixed) performed better. The primary reason for wavering performance in Ionosphere and Sonar datasets could be attributed to the large number of attributes in these datasets (Table 1). Due to curse of dimensionality, distance function looses its meaning by directly affecting nearest neigbours. Adding constraints do provide some contrast so as to group similar objects, but overall discernability is still less. It is important to note that, we did not search or tune for optimal values of (η, λ) for any particular dataset. During our initial investigation, we found that, for some change in values, the results were improved. We did some experiments on iris dataset and were able to achieve average Rand Index value of 0.99 and quite often achieved perfect clustering

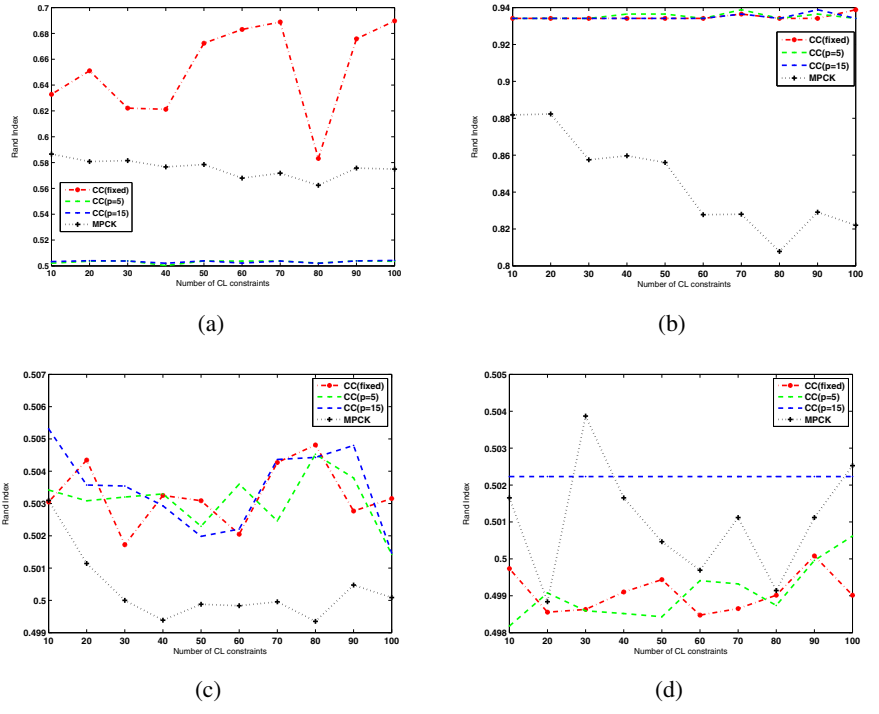

Fig. 2. Different versions of Constrained CHAMELEON(CC) compared with MPCK-means using Rand Index values averaged over 10 runs with CL constraints on different UCI datasets (a) Ionosphere (b) Iris (c) Liver (d) and Sonar

(Rand Index=1) during some of the runs for 190 constraints, with same settings as used in all other experiments shown. However, it will be too early to conclude that finding tuned values for (η, λ) will always increase performance, based on some initial results and will need further experimental evidence.

Based on our findings, we observed that changing values for (η, λ) did sometimes increase performance, but not consistently and can also sometimes lead to decrease in performance. We were also surprised by this phenomenon demonstrated by both the algorithms. In our case, carrying more experiments with additional constraints revealed that this decrease in performance is true upto a particular number of constraints. After that we again see rise in performance and *with enough number of constraints* (1% to 5% of constraints in our case with these datasets), *we are able to decipher original clustering* or close to it (Rand Index close to 1.0). CC(fixed) compared to other variants of CC were only slightly different on an average. CC(fixed) performed reasonably well across all the datasets on nearly all settings with MPCK-means. Other variants of CC were also better on an average compared to MPCK-means. Thus, *our algorithm performed better than MPCK-means in terms of handling the decrease in performance when the number of constraints increase*. Most importantly, *our algorithm performed well despite not trying to satisfy constraints* implicitly or explicitly.

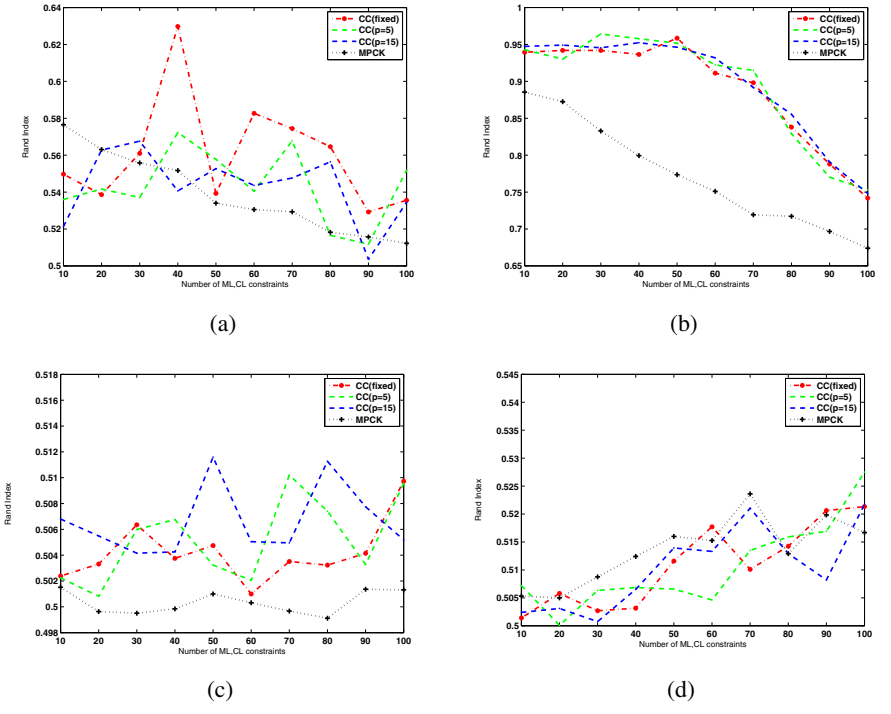

Fig. 3. Different versions of Constrained CHAMELEON(CC) compared with MPCK-means using Rand Index values averaged over 10 runs with ML and CL constraints on different UCI datasets (a) Ionosphere (b) Iris (c) Liver and (d) Sonar

6 Conclusion

In this work, we presented a novel constrained graph-based clustering method based on the CHAMELEON algorithm. We proposed a new framework for embedding constraints into the graph-based clustering algorithm to obtain promising results. Specifically, we thoroughly investigated the "how and when to add constraints" aspect of the problem. We also proposed a novel method for the distance limit criteria while embedding constraints into the distance function. Our algorithm outperformed the popular MPCK method on several real-world datasets under various constraint settings.

References

1. Basu, S., Banerjee, A., Mooney, R.J.: Semi-supervised clustering by seeding. In: Proceedings of the Nineteenth International Conference on Machine Learning (ICML 2002), pp. 27–34 (2002)
2. Basu, S., Bilenko, M., Mooney, R.J.: A probabilistic framework for semi-supervised clustering. In: Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 59–68 (2004)

3. Basu, S., Davidson, I., Wagstaff, K.: Constrained Clustering: Advances in Algorithms, Theory, and Applications. Chapman & Hall/CRC (2008)
4. Bilenko, M., Basu, S., Mooney, R.J.: Integrating constraints and metric learning in semi-supervised clustering. In: Proceedings of the Twenty-first International Conference on Machine Learning, ICML 2004 (2004)
5. Davidson, I., Ravi, S.S.: Agglomerative Hierarchical Clustering with Constraints: Theoretical and Empirical Results. In: Jorge, A.M., Torgo, L., Brazdil, P.B., Camacho, R., Gama, J. (eds.) PKDD 2005. LNCS (LNAI), vol. 3721, pp. 59–70. Springer, Heidelberg (2005)
6. Davidson, I., Ravi, S.S., Shamis, L.: A sat-based framework for efficient constrained clustering. In: Jonker, W., Petković, M. (eds.) SDM 2010. LNCS, vol. 6358, pp. 94–105. Springer, Heidelberg (2010)
7. Davidson, I., Wagstaff, K.L., Basu, S.: Measuring Constraint-Set Utility for Partitional Clustering Algorithms. In: Fürnkranz, J., Scheffer, T., Spiliopoulou, M. (eds.) PKDD 2006. LNCS (LNAI), vol. 4213, pp. 115–126. Springer, Heidelberg (2006)
8. Frank, A., Asuncion, A.: UCI machine learning repository (2010), http://archive.ics.uci.edu/ml
9. Gunopulos, D., Vazirgiannis, M., Halkidi, M.: From unsupervised to semi-supervised learning: Algorithms and evaluation approaches. In: SIAM International Conference on Data Mining: Tutorial (2006)
10. Halkidi, M., Gunopulos, D., Kumar, N., Vazirgiannis, M., Domeniconi, C.: A framework for semi-supervised learning based on subjective and objective clustering criteria. In: Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005), pp. 637–640 (2005)
11. Karypis, G., Han, E.-H., Kumar, V.: Chameleon: Hierarchical clustering using dynamic modeling. IEEE Computer 32(8), 68–75 (1999)
12. Karypis, G., Kumar, V.: Metis 4.0: Unstructured graph partitioning and sparse matrix ordering system. Tech. Report, Dept. of Computer Science, Univ. of Minnesota (1998)
13. Klein, D., Kamvar, S.D., Manning, C.D.: From instance-level constraints to space-level constraints: Making the most of prior knowledge in data clustering. In: Proceedings of the Nineteenth International Conference on Machine Learning (ICML 2002), pp. 307–314 (2002)
14. Kulis, B., Basu, S., Dhillon, I.S., Mooney, R.J.: Semi-supervised graph clustering: a kernel approach. In: Proceedings of the Twenty-Second International Conference on Machine Learning (ICML 2005), pp. 457–464 (2005)
15. Lelis, L., Sander, J.: Semi-supervised density-based clustering. In: Perner, P. (ed.) ICDM 2009. LNCS, vol. 5633, pp. 842–847. Springer, Heidelberg (2009)
16. Rand, W.M.: Objective criteria for the evaluation of clustering methods. Journal of the American Statistical Association 66(336), 846–850 (1971)
17. Ruiz, C., Spiliopoulou, M., Menasalvas, E.: Density based semi-supervised clustering. Data Mining and Knowledge Discovery 21(3), 345–370 (2009)
18. Tan, P.-N., Steinbach, M., Kumar, V.: Introduction to Data Mining, US edition. Addison Wesley, Reading (2005)
19. Wagstaff, K., Cardie, C., Rogers, S., Schrödl, S.: Constrained k-means clustering with background knowledge. In: Proceedings of the Eighteenth International Conference on Machine Learning (ICML 2001), pp. 577–584 (2001)