Efficient Hypergraph Clustering

Marius Leordeanu[1] **Cristian Sminchisescu**[2,1]
[1]Institute of Mathematics of the Romanian Academy
[2]Faculty of Mathematics and Natural Science, University of Bonn
marius.leordeanu@imar.ro, cristian.sminchisescu@ins.uni-bonn.de

Abstract

Data clustering is an essential problem in data mining, machine learning and computer vision. In this paper we present a novel method for the hypergraph clustering problem, in which second or higher order affinities between sets of data points are considered. Our algorithm has important theoretical properties, such as convergence and satisfaction of first order necessary optimality conditions. It is based on an efficient iterative procedure, which by updating the cluster membership of all points in parallel, is able to achieve state of the art results in very few steps. We outperform current hypergraph clustering methods especially in terms of computational speed, but also in terms of accuracy. Moreover, we show that our method could be successfully applied both to higher-order assignment problems and to image segmentation.

1 Introduction

Clustering is an essential problem in data mining, machine learning and computer vision [13]. Even though there is no definitive formulation of the clustering problem, it is generally believed that objects belonging to the same cluster should exhibit agreement relationships among each other, whereas objects that do not belong to a cluster should not exhibit such relationships. Many existing clustering methods are partition based [14, 22, 7, 19, 11]. They make the assumption that every data point belongs to a cluster. While they have good performance on problems where such assumptions are valid, in most real applications there is a large number of outliers that do not belong to any cluster. Deciding on whether a point is an outlier or not is a challenging task, which is why most partition based methods have poor performance when outliers abound, as also noted by other authors [17, 5].

Another limitation of most classical data clustering methods is their handling of only pairwise relationships between data points, given by the weights of an affinity graph. Since, in some applications, pairwise relations are not sufficient, it is important to develop algorithms that can handle higher-order relationships among data points. Since the problem becomes increasingly harder as the order increases, efficiency is a crucial factor.

In this paper we propose a novel and efficient method for clustering, that is not partition based and can handle higher-order relationships among sets of data points. Our algorithm has important theoretical properties such as convergence and satisfaction of first order necessary optimality conditions. In our experiments it significantly outperforms the current state of the art methods in terms of computational speed, while being at least as accurate.

Current research on hypergraph clustering takes on several directions. One is to transform the hypergraph into a graph by mapping the higher-order affinities to pairwise relationships [25, 2, 20]. Another direction is to generalize the methods from pairwise clustering, such as normalized cuts [22] and nonnegative matrix factorization to hypergraphs and their corresponding tensors [24, 21].

Our formulation is related to more recent hypergraph clustering methods [5] and [17]. In [5] the authors propose a formulation based on game theory and optimize the objective function using the results of Baum-Eagon growth transformation [3]. Their algorithm, like ours, iteratively updates the cluster membership for all nodes in parallel, and converges relatively fast. We show experimentally that our method, by taking larger steps towards a maximum, has significantly better speed of convergence, with slightly better accuracy. In turn, the algorithm of [17] performs node-wise up-

Appearing in Proceedings of the 15^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2012, La Palma, Canary Islands. Volume XX of JMLR: W&CP XX.

dates and converges slowly, and in our experiments is at least an order of magnitude slower than the method we propose.

2 Problem Formulation

We define clustering with k-th order relationships on a hypergraph, also known as a k-graph, following a formulation similar to [5, 17]. A k-graph $G = (V, E, w)$ is formed by a set of n vertices $V = \{1, ..., n\}$, a set of hyperedges $E \subseteq V^k$ and a k-th order real valued affinity function $w : W \rightarrow R$. The affinity function captures the strength of relationships on hyperedges, namely the stronger the affinity associated with a hyperedge $\{v_1, ..., v_k\}$, the larger the function value $w(\{v_1, ..., v_k\})$. We define the associated super-symmetric tensor W of the k-graph as follows: $W(v_1, ..., v_k) = w(\{v_1, ..., v_k\})$ if the hyperedge $\{v_1, ..., v_k\}$ is in E and 0 otherwise. The tensor W is super-symmetric since vertices within a given hyperedge can be considered in any order without changing the corresponding affinity value, and each hyperedge $\{v_1, ..., v_k\}$ has $k!$ duplicate entries in the tensor W.

Strong clusters $C \subseteq V$ are sets of vertices with high corresponding hyperedge affinities. Similar to [17] we describe the cluster score as the average over the hyperedge affinities within that cluster. Therefore, if a set C has m vertices, the cluster score can be written as:

$$S_C = \frac{1}{m^k} \sum_{v_1,...,v_k \in C} W(v_1, ..., v_k) \tag{1}$$

We define the vector $\mathbf{x}$, which acts as an indicator function, such that $x_i = 1/m$ if vertex i is in the cluster C and $x_i = 0$ otherwise. The cluster score can be rewritten as:

$$S(\mathbf{x}) = \sum_{v_1,...,v_k \in V} W(v_1, ..., v_k) \prod_{i=1}^{k} x_{v_i}. \tag{2}$$

Finding a good cluster means finding a subset of features with a high cluster score S_C. Even if we knew the number of elements in the cluster, maximizing S_C optimally would be an expensive combinatorial problem. For practical applications we want to find a solution efficiently, so we rely on approximations. We relax the problem, by allowing x to take values in the continuous domain $[0, \epsilon]$; ϵ acts as an upper bound of the cluster membership probability:

$$\mathbf{x}^* = \operatorname{argmax} S(\mathbf{x}) \;\text{ s.t. }\; \sum x_i = 1,\; \mathbf{x} \in [0, \epsilon]^n. \tag{3}$$

When $\epsilon = 1$, this hypergraph clustering formulation is equivalent to that of [5]. In the case of $\epsilon = 1$ and pairwise affinities in $\{0, 1\}$ (corresponding to unweighted graphs), the problem is identical to the classical computation of maximal cliques [8, 18].

The L1 norm constraint in Problem 3 favors in practice sparse solutions and biases towards categorical values for the membership assignments. This makes it easier to discretize and find actual clusters. Multiple clusters can be found in different ways, such as the ones proposed in [5, 17]. One idea [5] is to remove the points belonging to a cluster that was found and restart the problem on the remaining points. The other approach [17] is to start from different initial solutions that are close to different clusters and locally maximize the score (3).

3 Algorithm

Our proposed method (Algorithm 2) is an iterative procedure that, at each iteration t, approximates the higher order score $S(\mathbf{x_t})$ by its first-order Taylor expansion around the current solution $\mathbf{x_t}$. This transforms the higher order optimization problem into a sequence of linear programs, each defined in the neighborhood of the solutions $\mathbf{x_t}$, at each time step t. Note that the first-order approximations can be globally optimized efficiently on the continuous domain $\sum x_i = 1,\; \mathbf{x} \in [0, \epsilon]^n$. We successfully took a similar optimization approach to the problems of MAP inference, and graph and hypergraph matching [15, 16].

Before presenting our method we first introduce some notation. Given a possible solution $\mathbf{x}$ in the continuous domain, let vector $\mathbf{d}(\mathbf{x})$ be a function of $\mathbf{x}$, obtained by marginalizing the tensor $\mathbf{W}$ as follows:

$$d_i(\mathbf{x}) = \sum_{v_1,...,v_{k-1}} W(v_1, ..., v_{k-1}, i) \prod_{j=1}^{k-1} x_j. \tag{4}$$

We can now write the first-order Taylor approximation of the clustering score around solution $\mathbf{x_t}$ (Equation 2), in the following form:

$$S(\mathbf{x}) \approx (1 - k) S(\mathbf{x_t}) + k \mathbf{d}(\mathbf{x_t})^\top \mathbf{x}. \tag{5}$$

Maximizing the first-order approximation in the continuous domain of (3) results in the following linear program defined using the current solution $\mathbf{x_t}$:

$$\mathbf{y}^* = \operatorname{argmax} \mathbf{d}(\mathbf{x_t})^\top \mathbf{y} \;\text{ s.t. }\; \sum y_i = 1,\; \mathbf{y} \in [0, \epsilon]^n. \tag{6}$$

This linear program can be optimally solved using the following algorithm:

Algorithm 1 Optimize Problem 6.

$\mathbf{d} \leftarrow \mathbf{d}(\mathbf{x_t})$
$c \leftarrow \lfloor 1/\epsilon \rfloor$
Sort $\mathbf{d}$ in decreasing order $d_{i_1} \geq ... \geq d_{i_c} \geq ... \geq d_{i_n}$
$y_{i_l} \leftarrow \epsilon$, for all $l \leq c$
$y_{i_{c+1}} \leftarrow 1 - c\epsilon$
$y_{i_l} \leftarrow 0$, for all $l > c+1$
return $\mathbf{y}$

It is relatively easy to show that the above method returns a global optimum of (6). Once we obtain $\mathbf{y}^*$, we continue by searching for the global optimum of the original clustering score on the line segment between the current solution $\mathbf{x_t}$ and $\mathbf{y}^*$, an approach related to the classical Frank-Wolfe method [10]. If the hypergraph order, k, is less than or equal to 3, then the global optimum on the line segment can be computed in closed form since the score becomes a quadratic (or a cubic) one dimensional function. For higher order scores an efficient line search algorithm can be applied. Note that, since both $\mathbf{x_t}$ and $\mathbf{y}^*$ obey the constraints $\sum x_i = 1,\ \mathbf{x} \in [0, \epsilon]^n$, every sample on the line between the two also obeys the constraints, so the maximizer will also be in the domain.

Our algorithm can be summarized as follows, in pseudo-code:

Algorithm 2 Efficient Hypergraph Clustering.

Initialize $\mathbf{x_0}$, $t \leftarrow 0$
repeat
Step 1: $\mathbf{y}^* \leftarrow \text{argmax}\, \mathbf{d}(\mathbf{x_t})^\top \mathbf{y}$ s.t. $\sum y_i = 1,\ \mathbf{y} \in [0, \epsilon]^n$. If $\mathbf{d}(\mathbf{x_t})^\top (\mathbf{y} - \mathbf{x_t}) = 0$ stop.
Step 2: $\alpha^* \leftarrow \text{argmax}\, S((1-\alpha)\mathbf{x_t} + \alpha \mathbf{y}^*),\ \alpha \in [0, 1]$.
Step 3: $\mathbf{x_{t+1}} \leftarrow (1 - \alpha^*)\mathbf{x_t} + \alpha^* \mathbf{y}$, $t \leftarrow t + 1$
until convergence
return $\mathbf{x_t}$

4 Theoretical Analysis

Proposition 1 The score $S(\mathbf{x}_t)$ increases at every step t of Algorithm 2 and the sequence $\mathbf{x}_t$ converges.

Proof: The algorithm does not stop at Step 2 if there is a $\mathbf{y}$ different from $\mathbf{x_t}$ such that $\mathbf{d}(\mathbf{x_t})^\top (\mathbf{y} - \mathbf{x_t}) > 0$. Since $\mathbf{d}(\mathbf{x_t})$ is proportional to the gradient of the original clustering score S at the current $\mathbf{x_t}$, it follows that there exists a point on the line segment between $\mathbf{x_t}$ and $\mathbf{y}$ with a score greater than $S(\mathbf{x_t})$. Such a point is found in Step 3 during line search (maximizer). Therefore, the score increases at every step. Since S is also bounded above, the algorithm will converge to a limit score. This must happen in the limit at step 2, when $\mathbf{d}(\mathbf{x_t})^\top (\mathbf{y} - \mathbf{x_t}) = 0$ and the solution $\mathbf{x_t}$ is returned.

Proposition 2: For a point of convergence $\mathbf{x}$, if $x_i < x_j$ then $d_i(\mathbf{x}) \leq d_j(\mathbf{x})$.

Proof: We will use a proof by contradiction. Let us assume that there exist i, j such that $x_i < x_j$ and $d_i(\mathbf{x}) > d_j(\mathbf{x})$. Let $r = (x_j - x_i)/2 > 0$ and $\mathbf{y}$ be a vector of the same size as $\mathbf{x}$, having all elements equal to those of $\mathbf{x}$ except for the i-th and j-th elements, which are $y_i = x_i + r$ and $y_j = x_j - r$. It can be easily verified that $\mathbf{y}$ lies in the valid continuous domain, since $\mathbf{x}$ is also in the domain. It can also be verified that $\mathbf{d}(\mathbf{x})^\top \mathbf{y} - \mathbf{d}(\mathbf{x})^\top \mathbf{x} = \mathbf{d}(\mathbf{x})^\top (\mathbf{y} - \mathbf{x}) = r(d_i(\mathbf{x}) - d_j(\mathbf{x})) > 0$, which contradicts the assumption that $\mathbf{x}$ is a point of convergence, since there exists at least one $\mathbf{y}$ which gives a better result in Step 2 of the algorithm. Hence, $\mathbf{x}$ cannot be a maximizer at Step 2.

Proposition 3 The points of convergence of Algorithm 2 satisfy the Karush-Kuhn-Tucker (KKT) necessary optimality conditions for Problem 3.

Proof:

The Lagrangian function of (3) is:

$$L(\mathbf{x}, \lambda, \mu, \beta) = S(\mathbf{x}) - \lambda(\sum x_i - 1) + \sum \mu_i x_i + \sum \beta_i (\epsilon - x_i), \tag{7}$$

where $\beta_i \geq 0$, $\mu_i \geq 0$ and λ are the Lagrange multipliers. Since $k d_i(\mathbf{x})$ is the partial derivative of $S(\mathbf{x})$ w.r.t x_i, the KKT conditions at a point $\mathbf{x}$ are:

$$\begin{array}{l} k d_i(\mathbf{x}) - \lambda + \mu_i - \beta_i = 0. \\ \sum_{i=1}^n \mu_i x_i = 0. \\ \sum_{i=1}^n \beta_i (\epsilon - x_i) = 0. \end{array}$$

As the elements of $\mathbf{x}$ and the Lagrange multipliers are non-negative, it follows that if $x_i > 0 \Rightarrow \mu_i = 0$ and $x_i < \epsilon \Rightarrow \beta_i = 0$. Then there exists a constant $\delta = \lambda / k$ such that the KKT conditions can be rewritten as:

$$d_i(\mathbf{x}) \begin{cases} \leq \delta, & x_i = 0, \\ = \delta, & x_i \in (0, \epsilon), \\ \geq \delta, & x_i = \epsilon. \end{cases}$$

If these KKT conditions are not met then the conclusion of Proposition 2 does not hold, implying that $\mathbf{x}$ is not a point of convergence, which gives a contradiction. Therefore, Proposition 3 must be true.

4.1 Computational Complexity

In the general case, when the number of hyperedges is of order $O(N^k)$ (N - number of data points, k the order of hyperedges) the overall complexity of each iteration of our algorithm is also $O(N^k)$ (linear in the number of hyperedges). It is important to note that [17, 5] have the same $O(N^k)$ complexity per step.

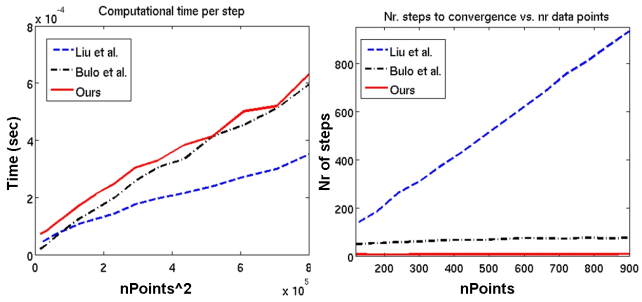

Figure 1: Average computational cost over 100 different experiments as a function of the problem size. Left: the linear dependency of the run time vs $nPoints^2$ shows experimentally that all algorithms have $O(N^2)$ complexity for hyperedges of order 2. Right: the method of [17] converges in a number of iterations that is directly proportional to the problem size, while the convergence of ours and [5] is relatively stable w.r.t. the number of points. In particular, ours reaches a high objective score in about 10 iterations.

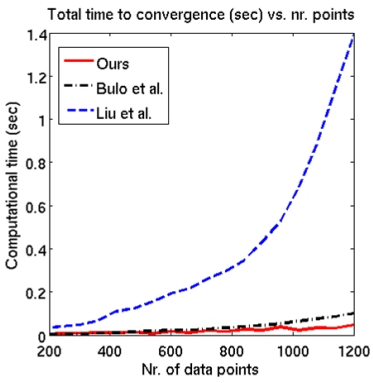

Figure 2: Average total time to convergence over 100 experiments. We let ours run for 50 iterations, but it usually converges after about 10 iterations.

In Figures 1 and 2 we present an evaluation of the computational costs of the three methods on synthetic problems with hyperedges of second order (k = 2). The results are averages over 100 different synthetic problems with varying number of data points. In Figure 1, left plot, we show the average computational time per iteration (in Matlab) of each method vs. the squared number of data points. Note that the computation time per step for each method varies almost linearly with the squared number of data points, confirming the theoretical $O(N^k)$ complexity (for $k = 2$). As expected, the least expensive method per step is [17] which performs sequential updates, while ours and [5] are based on parallel updates. The drawback of [17] (right plot) is the large number of iterations to convergence. It is interesting to note that [17] needs approximately the same number of iterations to converge as the dimensionality of the problem (number of data points). On the other hand, both ours and [5] are relatively stable with respect to the number of points. Ours converges the fastest, needing on average less than 10 iterations. In terms of overall computational time, the method of [17] takes 1 to 2 orders of magnitude longer than ours to converge for $200 - 1200$ points (Figure 2).

5 Experimental Analysis

We present three types of experiments that are relevant for data clustering, and compare the performance of our algorithm with that of current state of the art methods. The first experiment is on 2D line fitting, similar to experiments from [17] and [5]. The second type of experiments we show is on higher order matching, where we perform comparisons with both clustering methods as well as two state of the art hypergraph matching algorithms [23, 9]. This experiment reveals an interesting connection between hypergraph clustering and hypergraph matching. The third set of experiments is on image segmentation, in which we emphasize the limitation of [5] to using nonnegative affinities, as opposed to ours and [17].

5.1 Line clustering

This experiment consists of finding lines as clusters of 2D points. Since any two points lie on a line, third order affinity measures are needed. Similar to [17] and [5], for any triplet of points $\{i, j, k\}$, we use as dissimilarity measure the mean distance to the best fitting line $d(i, j, k)$. We define the similarity function $w(\{i, j, k\})$ using a Gaussian kernel $w(\{i, j, k\}) = exp(-d(i, j, k)^2/\sigma_d^2)$, where σ_d is a parameter that controls the sensitivity to fitting errors.

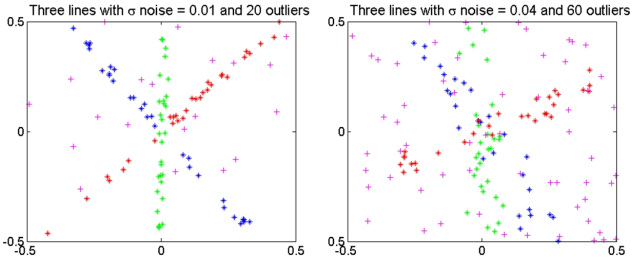

Figure 3: Generated 3D points from three lines with outliers. Notice the difficulty of finding the correct clusters in the example on the right, when both the noise variance and the number of outliers are relatively large. Best viewed in color.

We randomly generate points belonging to three lines in the range $[-0.5, 0.5]^2$, with 30 points per line perturbed with Gaussian noise $N(0, \sigma)$ and add a number of outliers in the same region of the 2D space. In Figure 3 we present two examples of generated point sets, with the three lines shown in red, blue and green and outliers shown with magenta. The figure on the right presents the case of 60 outliers with a relatively large amount of noise $\sigma = 0.04$. Note the difficulty of recovering the correct clusters.

We are interested in two important aspects: accuracy and speed of convergence, under varying levels of noise and number of outliers. For each level of noise and number of outliers, we test all methods in 50 random trials and average the results (Figure 4). For each clustering problem, all algorithms are initialized with the same uniform initial solution, and the performance is evaluated using the F-measure. Our method and [17] use the same $\epsilon = 1/30$. The performance of all methods in our experiments was relatively stable w.r.t σ_d, so we fixed $\sigma_d = 0.02$.

In Figure 4, plots b and d, we show the average F-measure of all three algorithms at convergence, ours performing slightly better then the rest. An important aspect in practice is how fast a method can reach a good solution. This issue is particularly important in graph and hypergraph clustering problems that have combinatorial complexity. Our algorithm obtains accurate solutions much faster than both [17] and [5]. If our method takes on average between 5 to 10 steps to reach a good solution, the algorithm of [17] needs more than 150 steps ($\approx$ the number of data points). Since the computation time per step can be shown to be comparable for all algorithms (ours is about 1.4 times slower per step than the others on these problems) we present in Figure 4, plots a and c, the performance of all algorithms when ours is stopped after 5 iterations, whereas the other two methods are stopped after 10 steps. Empirically we find that our proposed method converges about one order of magnitude faster.

In Figure 5 we further study the convergence properties of all three algorithms. Plots a, b and c show the solutions on one experiment for all methods after 5, 50 iterations and at convergence. The case presented is a good example of how the methods behave in general. Our method, after only 5 iterations, is already close to its optimum, reaching a sparse solution. Note that the first 30 elements correspond to the points of the first line, so our method is already close to a perfect solution. The method of [17] is still very close to its initial solution, due to its site-wise updates. The method of [5] based on Baum-Eagon [3] performs parallel updates per step, but it takes smaller steps than ours. Our method uses the same problem formulation as [17] so we directly compare the objective scores obtained

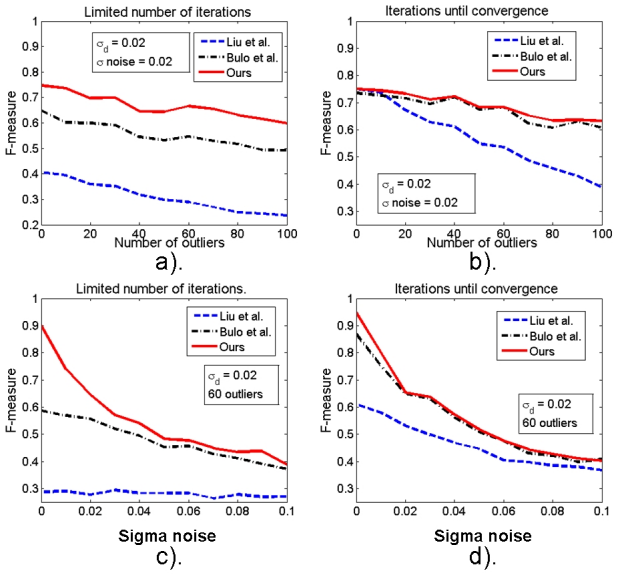

Figure 4: Performance comparison of different methods on 2D line fitting. Left: our method has a far superior performance when all algorithms are allowed to run for the same, limited amount of computational time: ours for 5 iterations (less time), the competitors for 10 iterations. Comparison with right plot: notice that our method has almost converged in only 5 iterations. Best viewed in color.

per iteration averaged over 300 experiments (Figure 5, plot d: average scores normalized by the maximum average). Our algorithm converges very fast (< 10 iterations), while [17] takes almost 200 steps until it converges to an objective score slightly less on average than ours.

5.2 Affine-invariant Matching

Affine invariant point matching is an important problem in computer vision, with applications to object matching and recognition. Distant or planar objects seen from different view-points undergo transformations that can be well approximated by an affine or a piece-wise affine transformation. We perform experiments on affine invariant point matching and compare our method with hypergraph clustering as well as hypergraph matching methods [9] (TM), [23] (PM). Since the connection between hypergraph clustering and hypergraph matching is not well established yet, we believe that such experiments are relevant.

Hypergraph matching has a similar tensor formulation as hypergraph clustering (3), the main difference being the constraints on the solution. Usually assignment problems impose 1-to-1 matching constraints on a solution $\mathbf{x}$, where each element x_i corresponds to a candidate match $i = (u, v)$. Here, u is the index of a

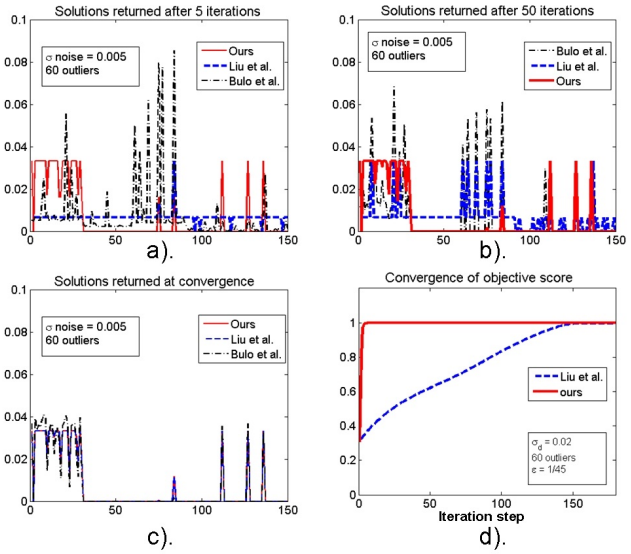

Figure 5: Solutions achieved after different numbers of iterations on the same problem for all algorithms. For plots a-c the x-axis indexes the elements of the solution vectors. Plot d: average objective scores of our method vs. [17] per iteration number. Notice the fast convergence of our method. Best viewed in color.

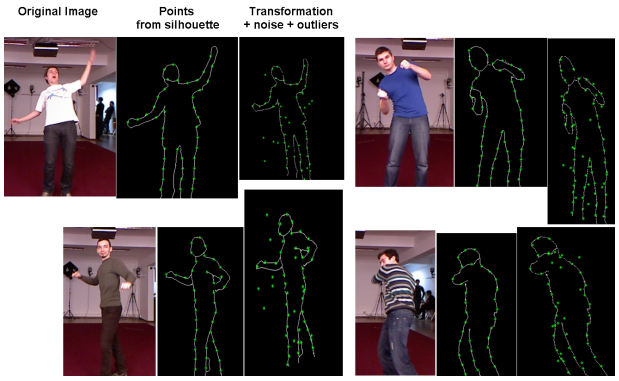

Figure 6: Examples from our dataset: original image (left), points sampled on the boundaries of the person (middle), transformed points with added noise and outliers (right). Best viewed in color

feature from one image and v is the index of the feature from the other image. The acceptable solutions are indicator vectors such that x_i is 1 if u is matched to v and 0 otherwise. A tensor, similar to W is constructed such that its elements of type $(i_1, ..., i_k)$ represent matching similarities between the k-tuple of features $(u_1, ..., u_k)$ from one image and the corresponding k-tuple $(v_1, ..., v_k)$ from the other image, where $i_q = (u_q, v_q)$.

Although the constraints are different for hypergraph matching, it is important to note that the two state-of-the-art hypergraph matching methods [9, 23] ignore the 1-to-1 constraints during optimization and impose them only during a final, post-processing step. This makes them very efficient in practice. The method of [9] is based on computing the stationary point of the higher order power method applied to the tensor W. That is the higher-order extension of the principal eigenvector of matrices, well known for its usefulness in clustering. We therefore expect other clustering methods to be useful for matching.

Since affine transformations cannot be recovered from pairs of points, we use third order clustering. Given three points (u_1, u_2, u_3) in one image and their corresponding matches (v_1, v_2, v_3) in the other image, transformed by the affine transformation $\mathbf{T}$, the areas of the corresponding triangles A_{u_1,u_2,u_3} and A_{v_1,v_2,v_3} are related by the formula $|\det \mathbf{T}| = A_{u_1,u_2,u_3}/A_{v_1,v_2,v_3}$. We use third order matching scores of the form $W(i_1, ..., i_k) = \exp\left((1 - \sqrt{|\det \mathbf{T}|A_{v_1,v_2,v_3}/A_{u_1,u_2,u_3}})^2/\sigma^2\right)$.

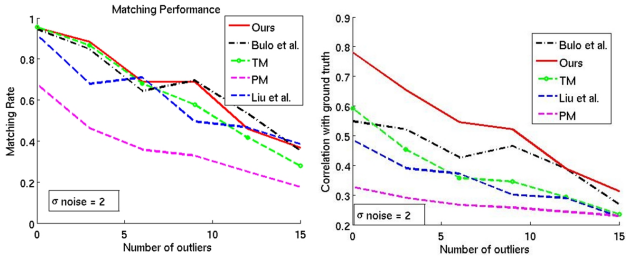

Figure 7: Results on affine invariant matching. Note that our method gives solutions that are already close to the sparse ground truth without using the 1-to-1 matching constraints. Best viewed in color

We generated 2D point sets from 420 real images containing 7 different persons collected with an RGB-depth kinect camera. For each image we first obtained the occlusion boundaries of the person and extracted around $40 - 50$ points per image equally spaced on the boundary. Using this set of points we obtained the second set by transforming the points based on a randomly generated affine transformation $\mathbf{T} = [1 + N(0, 0.2), N(0, 0.2); N(0, 0.2), 1 + N(0, 0.2)]$, and then perturbed their position with Gaussian noise $N(0, \sigma)$. We also added outliers to the second set of points (Figure 6). Since $|\det \mathbf{T}|$ is unknown at testing time, we estimated its distribution (from the distribution of $\mathbf{T}$), and for each matching example, we sampled a few values and picked the one giving the best objective score.

In clustering, a sparse solution is always preferred. In the case of matching, in order to obtain a sparse

solution it is better to use the 1-to-1 matching constraints. If clustering already provides a sparse solution it is not clear how to use the 1-to-1 constraints in the final post-processing step. The three clustering methods from the previous experiment tend to give sparse solutions. In turn, hypergraph matching algorithms [9, 23] almost always give continuous solutions. They could be later discretized by the Hungarian algorithm that imposes the 1-to-1 constraints. We therefore expect the matching algorithms to benefit more from the Hungarian method than the clustering algorithms. For this reason we evaluate algorithms in two ways. First, we measure the matching rate by applying the Hungarian method to the raw output $\mathbf{x}$ of each algorithm , namely solve the problem $\mathbf{y}^* = \text{argmax}\, \mathbf{x}^\top \mathbf{y}$, given the 1-to-1 matching constraints. Second, we measure the closeness of the raw output to the sparse ground truth, by computing the normalized correlation between the two. Note that the second measure should favor methods that give sparse clustering solutions, which are close to the matching ground truth without the knowledge of the one-to-one constraints. We present the results in Figure 7. Note that our method outperforms on average all the others in terms of both matching and normalized correlation scores against ground truth. The fact that our method gives sparse solutions close to ground truth, without using the 1-to-1 matching constraints could be useful in matching applications where the 1-to-1 constraints do not hold. For example, in cases of large changes in scale or when matching sub-parts of objects to full parts, the exact many-to-1 or many-to-many matching constraints are unknown. In those cases, a robust clustering method could recover the correct matches without using any matching constraints.

5.3 Image Segmentation

A recent trend in image segmentation is to formulate the problem in terms of composition [6, 12, 4] from a bag of single segments, obtained from running multiple segmentation algorithms. The composition of segments becomes, at a higher level, another clustering problem. In this case, each node of the graph is a possible segment (one image region or object), and affinities between nodes represent relationships between image segments. The problem is usually formulated as a pairwise clustering problem, with unary and pairwise affinities/costs. The unary terms represent the "goodness" of single image segments, and the pairwise terms capture how well neighboring segments agree, based on their common contours or other features [12]. When two segments overlap there is a conflict, which is captured by repulsion terms (negative affinities). A final segmentation (tiling of individual segments) should contain a set of segments that cover the entire image, while no two segments overlap.

Segmentation is interesting from a clustering point of view, due to the presence of negative pairwise terms (conflicts), which are not common to clustering methods. Our method and [17] can accommodate negative terms, whereas the method of [5] is based on the Baum-Eagon growth transformation for polynomials with non-negative coefficients. Our experiments on segmentation confirm that [5] is not competitive when negative affinities exist.

In theory [5] could be applied to problems with negative affinities, if a positive constant is added to each term, such that all become non-negative. The addition of a constant does not change the optimum of the original problem, but it does affect the optimization of the relaxed problem.

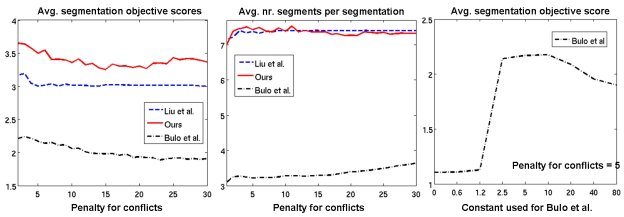

Figure 9: Quantitative segmentation results. Left: Average objective scores after removing segments with conflicts. Middle: average number of surviving segments in the final tiling, after eliminating conflicts. Right: average objective score for [5] when different constants are added, for conflict penalty = 5.

We perform clustering experiments for segmentation on the BSDS300 dataset [1] using the nodes (segments) and the affinities (unary and pairwise) provided by the authors of [6, 12] (there are between 10 to 300 possible segments for each image). For each segment i the unary term $W(i,i)$ is positive and increases with the quality of the segment. For every pair of segments $W(i,j)$ is positive if the segments are neighbors and agree with each other, negative (minus a large penalty constant) when the segments overlap, and 0 otherwise. The problem consists of finding a set of segments which maximizes the clustering score.

For each image the final number of segments should be the same as the ground truth (we consider the minimum number from the several human segmentations given in [1]). For each method, we sort the elements of the continuous solution in decreasing order and keep the first N_s segments (where N_s = ground truth). If the tiling obtained contains segments that are in conflict, we remove segments in a greedy manner, starting with the one with the largest number of conflicts, until all conflicts are eliminated. The same greedy procedure is applied for all methods. In Figure 9 (left) we show the average objective score (over the 100 im-

Figure 8: Examples of segmentation results after removing segments that have conflicts. Dark regions are not covered by any segment. Note the better image coverage and quality of the segmentations returned by our method. Best viewed in color.

ages from the training set) as we vary the value of the penalty (subtracted for pairs of segments that are in conflict). For [5] we added a positive constant equal to this penalty to every affinity term during optimization, and subtracted it back for evaluating the final score. We experimented with different values of the added constant for [5], and the optimum was always close to the value of the penalty (Figure 9, right plot). The middle plot shows the average number of segments that survive after removing the ones in conflict. Before the removal procedure our method also outperformed (w.r.t. objective score) [17] on 62 images and [5] on 98 images out of 100. In Figure 8 we present a few representative segmentation results. The dark regions are not covered by any segment. Note that another post-processing greedy procedure could be used to fill the uncovered areas. Quantitatively, ours has on average the best performance, covering the images better and giving higher scores. On average both our method and [17] found a cluster of segments with no conflicts in about 60% of cases, while [5] did so only in less than 10%. These are cases in which the greedy removal procedure was not necessary.

6 Conclusions

We presented a novel hypegraph clustering method with important theoretical properties and state of the art performance. Our algorithm is based on an efficient iterative procedure with significantly better convergence speed than existing state of the art methods, without any loss of accuracy. We have also tested our method on matching and image segmentation problems and showed competitive performance. In future work we plan to further explore the application of hypergraph clustering techniques to matching, visual recognition and segmentation problems.

Acknowledgments

This work was supported by CNCSIS-UEFISCDI, under PNII-RU-RC-2/2009, and the EC, under MCEXT-025481.

References

[1] http://www.eecs.berkeley.edu/research/projects/cs.

[2] S. Agarwal, J. Lim, L. Zelnik-Manor, P. Perona, D. Kriegman, and S. Belongie. Beyond pairwise clustering. In *International Conference on Computer Vision and Pattern Recognition*, 2005.

[3] L. Baum and J. Eagon. An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology. In *Bull. Amer. Math. Soc*, 1967.

[4] W. Brendel and S. Todorovic. Segmentation as maximum-weight independent set. In *Neural Information Processing Systems*, 2010.

[5] S. Bulo and M. Pellilo. A game-theoretic approach to hypergraph clustering. In *Neural Information Processing Systems*, 2009.

[6] J. Carreira and C. Sminchisescu. Constrained parametric min-cuts for automatic object segmentation. In *IEEE International Conference on Computer Vision and Pattern Recognition*, 2010.

[7] I. Dhillon, Y. Guan, and B. Kulis. Kernel k-means: spectral clustering and normalized cuts. In *ACM International Conference on Knowledge Discovery and Data Mining*, 2004.

[8] C. Ding, T. Li, and M. Jordan. Nonnegative matrix factorization of combinatorial optimization: Spectral clustering, graph matching, and clique finding. In *IEEE International Conference on Data Mining*, 2008.

[9] O. Duchenne, F. Bach, I. Kweon, and J. Ponce. A tensor-based algorithm for high-order graph matching. In *International Conference on Computer Vision and Pattern Recognition*, 2009.

[10] M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.

[11] B. Frey and D. Dueck. Clustering by passing messages between data points. In *Science*, 2007.

[12] A. Ion, J. Carreira, and C. Sminchisescu. Image segmentation by figure-ground composition into maximal cliques. In *IEEE International Conference on Computer Vision*, 2011.

[13] A. Jain, M. Murty, and P. Flynn. Data clustering: a review. In *ACM Computing Surveys*, 1999.

[14] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu. An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology. In *Pattern Analysis and Machine Intelligence*, 2002.

[15] M. Leordeanu, M. Hebert, and R. Sukthankar. An integer projected fixed point method for graph matching and map inference. In *Neural Information Processing Systems*, 2009.

[16] M. Leordeanu, A. Zanfir, and C. Sminchisescu. Semi-supervised learning and optimization for hypergraph matching. In *IEEE International Conference on Computer Vision*, 2011.

[17] H. Liu, L. Latecki, and S. Yan. Robust clustering as ensembles of affinity relations. In *Neural Information Processing Systems*, 2010.

[18] T. Motzkin and E. Straus. Maxima for graphs and a new proof of a theorem of turan. In *Canad. J. Math.*, 1965.

[19] A. Ng, M. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. In *Neural Information Processing Systems*, 2002.

[20] J. Rodriguez. On the laplacian spectrum and walk-regular hypergraphs. In *Linear and Multilinear Algebra*, 2003.

[21] A. Shashua, R. Zass, and T. Hazan. Multi-way clustering using super-symmetric non-negative tensor factorization. In *European Conference on Computer Vision*, 2003.

[22] J. Shi and J. Malik. Normalized cuts and image segmentation. *Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[23] R. Zass and A. Shashua. Probabilistic graph and hypergraph matching. In *International Conference on Computer Vision and Pattern Recognition*, 2008.

[24] D. Zhou, J. Huang, and B. Scholkopf. Learning with hypergraphs: clustering, classification and embedding. In *Neural Information Processing Systems*, 2007.

[25] J. Zien, M. Schlag, and P. Chan. Multilevel spectral hypergraph partitioning with arbitrary vertex sizes. In *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, 1999.