An Incremental Reseeding Strategy for Clustering

Xavier Bresson* Huiyi Hu† Thomas Laurent‡ Arthur Szlam§
James von Brecht¶

Abstract

We propose an easy-to-implement and highly parallelizable algorithm for multiway graph partitioning. The algorithm proceeds by alternating three simple routines in an iterative fashion: diffusion, thresholding, and random sampling. We demonstrate experimentally that the proper combination of these ingredients leads to an algorithm that achieves state-of-the-art performance in terms of cluster purity on standard benchmark data sets. Moreover, the algorithm runs an order of magnitude faster than the other algorithms that achieve comparable results in terms of accuracy.

1 Introduction

One of the most basic unsupervised learning tasks is to automatically partition data into clusters based on similarity. A standard scenario is that the data are represented as a weighted graph. Data points correspond to vertices on the graph while edges between vertices encode the similarity between data points. Many of the most popular and widely used clustering algorithms, such as spectral clustering, fall into this category. Despite the vast literature on graph-based clustering, the field remains an active area for both theoretical and practical research.

In this work, we propose a resampling-based spectral algorithm for multiway graph partitioning. On graphs that contain reasonably well-balanced clusters of medium scale, the algorithm provides a strong combination of accuracy, efficiency and robustness to noise in the graph construction process. The algorithm is also exceedingly simple, intuitive and trivial to implement (the appendix provides a 32-line MATLAB code). It also parallelizes trivially, and can therefore scale gracefully to large numbers of clusters as well as to graphs with large numbers of vertices.

We validate these claims via an extensive experimental evaluation of the algorithm. We exhaustively test our algorithm on a total of 26 real world data sets. We also provide a detailed algorithmic comparison using four recent clustering algorithms that claim state-of-the-art results. These experiments demonstrate that our algorithm achieves state-of-the-art performance in terms of cluster purity while running an order of magnitude faster than the other highly accurate clustering methods (e.g. [1]) that we compare against. Moreover, our strategy performs well using only random initialization. This contrasts with other state-of-the-art methods that rely on a lower-level algorithm (such as NCut) for initialization. We also provide experiments to demonstrate the robustness of the algorithm with respect to noise and perturbations in the underlying graph. While many highly accurate algorithms exhibit a sharp decrease in accuracy if the input graph is corrupted by noise, our algorithm remains stable: the accuracy of our algorithm decays slowly and gracefully with increasing levels of noise. These results, when taken together, lead to an algorithm with a quite appealing combination of simplicity, performance and ease in out-of-the-box usage.

*École polytechnique fédérale de Lausanne, (xavier.bresson@epfl.ch)
†Department of Mathematics, University of California, (huiyihu@math.ucla.edu)
‡Department of Mathematics, Loyola Marymount University, (tlaurent@lmu.edu)
§Facebook, (aszlam@fb.com)
¶Department of Mathematics, California State University, Long Beach (James.vonBrecht@csulb.edu)

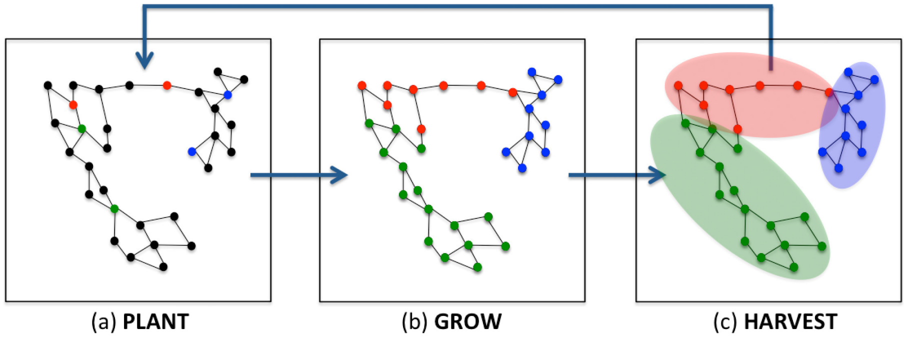


Figure 1: Illustration of the Incremental Reseeding (INCRES) Algorithm for $R = 3$ clusters. The various colors red, blue, and green identify the clusters. Figure (a): At this stage of the algorithm, $s = 2$ seeds are randomly planted in the clusters computed from the previous iteration. Figure (b): The seeds grow with the random walk operator. Figure (c): A new partition of the graph is obtained and used to plant $s + ds$ seeds into these clusters at the next iteration.

2 Description of the Algorithm

The main idea behind our algorithm arises from a well-known and widely used property of the random walk on a graph. Specifically, a random walker started in a low conductance cluster is unlikely to leave that cluster quickly [2]. This fact provides the basis for transductive label propagation methods [3] as well as for "local" clustering methods [4]. In label propagation, for instance, an oracle provides a set of labeled vertices that are propagated along the graph using a random walk matrix or a diffusion matrix. Each unlabeled vertex is then associated to the label which, after being propagated, best represents the given unlabeled vertex.

Our algorithm simply iterates upon this basic idea. Assume that the graph has R well-defined clusters of comparable size and low conductance. If we knew these clusters in advance, we could then select a handful of "seed" vertices in the center of each cluster. We would then expect to obtain good results from a transductive label propagation by using these seeds as labels. In an unsupervised context we cannot, of course, *a-priori* place seeds in the center of each cluster. To overcome this, we instead place a handful of seeds at random. We then apply a random walk matrix or diffusion matrix a few times to propagate these seeds. We finally obtain a temporary clustering by assigning each vertex to the seed which, after propagation, best represents the vertex. We then choose new seeds from these temporary clusters and iterate the process. If the clusters improve then the seeds will likely improve, and vice-versa. This incites a feed-back loop and we get a virtuous cycle. We can then excite the speed and improve the quality of this cycle by gradually drawing more and more seeds throughout the process. We refer to this idea as an *incremental reseeding strategy*. Figure 1 depicts this cyclic process graphically.

2.1 The Basic Algorithm

To formalize these ideas, let $G = (V, W)$ denote a weighted, connected graph on N vertices $V = \{v_1, \ldots, v_N\}$ with edge weights $W = \{W_{ij}\}_{i,j=1}^{N}$ that encode a measure of similarity between each pair (i, j) of vertices. Let D denote the diagonal matrix of (weighted) vertex degrees. The algorithm starts from a random partition $\mathcal{P} = (\mathcal{C}_1, \ldots, \mathcal{C}_R)$, $\mathcal{C}_1 \cup \cdots \cup \mathcal{C}_R = V$, $\mathcal{C}_r \cap \mathcal{C}_q = \emptyset$ $(r \neq q)$ of the vertices. In other words, each v_i is assigned to one of the R clusters uniformly at random. Let $s = 1$ denote the initial number of seeds. At each of the successive iteration, we update the current partition $\mathcal{P} = (\mathcal{C}_1, \ldots, \mathcal{C}_R)$ according to the steps outlined in the pseudocode for the INCRES Algorithm. The pseudocode for the INCRES Subroutines appear on the subsequent page.

At the beginning of each iteration, the routine PLANT($\mathcal{P}$,s) will sample s seeds from each of the R clusters $\mathcal{C}_r$ in the current partition $\mathcal{P}$ uniformly at random. These Rs seeds furnish the temporary labels that the GROW routine then propagates along the graph using a random walk matrix. We initialize GROW with an $N \times R$

INCRES Algorithm

Input: Similarity matrix W, seed increment ds, number of clusters R.
Initialization: $s = 1$, random partition $\mathcal{P}$.
repeat
  $F =$ PLANT$(\mathcal{P}, s)$
  $F \leftarrow$ GROW(F, W)
  $\mathcal{P} \leftarrow$ HARVEST(F)
  $s \leftarrow s + ds$
until $\mathcal{P}$ converges
Output: $\mathcal{P}$

matrix F, where $F_{i,r} = 1$ if vertex v_i was drawn from cluster $\mathcal{C}_r$ and $F_{i,r} = 0$ otherwise. We then iteratively apply the random walk transition matrix WD^{-1} to F until each vertex has a nonzero probability of being visited by a random walker, i.e. until each entry $F_{i,r}$ is nonzero. Finally, the routine HARVEST simply assigns each vertex v_i to its most likely cluster, or in other words the cluster for which $F_{i,r}$ is maximal. This produces a new partition $\mathcal{P}$ of the vertices into R clusters. We then increment s to $s + ds$, and use this partition and number of seeds s to initialize PLANT at the beginning of the next iteration. We refer to this overall procedure as the Incremental Reseeding Algorithm (INCRES).

INCRES Subroutines

function PLANT$(\mathcal{P}, s)$
Initialize F as an N-by-R matrix of zeros.
for $r = 1$ to R **do**
  for $k = 1$ to s **do**
    Draw at random a vertex i in cluster $\mathcal{C}_r$.
    $F_{i,r} \leftarrow F_{i,r} + 1$
  end for
end for
return F.
end function

function GROW(F, W)
while $\min_i \min_r F_{i,r} = 0$ **do**
  $F \leftarrow \left(WD^{-1}\right) F$
end while
return F.
end function

function HARVEST(F)
for $r = 1$ to R **do**
  $\mathcal{C}_r = \{i : F_{i,r} \geq F_{i,s} \text{ for all } s\}$
end for
return $\mathcal{P} = (\mathcal{C}_1, \ldots, \mathcal{C}_R)$
end function

The overall routine has only a single parameter ds that controls the linear rate at which the number of seeds drawn at each iteration increases. In practice, we select

$$ds = \textbf{speed} \times 10^{-4} \times \frac{N}{R} \tag{1}$$

for some proportionality constant **speed** between one and ten. By rescaling ds in this way, a constant of proportionality **speed** $= 1$ corresponds to a total of $s = 0.1N/R$ seeds planted in each cluster after 1000 iterations. Assuming well-balanced clusters of roughly equal size, approximately one-tenth of each cluster is sampled after 1000 iterations. Drawing a significant fraction of each cluster will cause the subsequent clustering to stabilize, leading to eventual convergence of the algorithm. Using around 10% of labels in each cluster is a typical level at which INCRES will stabilize.

The parameter ds therefore represents a "timestep" for INCRES, and the overall algorithm behaves well with respect to this parameter. In general, small increments ds will lead to slower convergence at higher accuracy while larger increments ds will lead to faster convergence but potentially less accurate solutions. Our experiments show that **speed** $= 1$ works remarkably well for a large variety of data sets. We also provide results for **speed** $= 5$, which yields faster stabilization with slightly less accuracy, to show that the algorithm is indeed robust and predictable with respect to the choice of this parameter.

The general INCRES framework also proves robust to implementation choices for the three main routines. For instance, in addition to the random walk matrix WD^{-1}, there exists a variety of alternative means to propagate labels along a graph. By-and-large, the overall INCRES strategy does not depend heavily upon the particular implementation of GROW, so long as it realizes the basic idea of label propagation in one form or another. For instance, we have found that replacing the random walk step $F \leftarrow WD^{-1}F$ with a diffusion step $F \leftarrow D^{-1}WF$ or $F \leftarrow D^{-1/2}WD^{-1/2}F$ will give similar results in many circumstances. Occasionally, we have found that utilizing a "personalized Page-Rank" step

$$F \leftarrow \alpha WD^{-1}F + (1-\alpha)F_0 \tag{2}$$

can give better performance on small data sets that contain a large (relative to the size of the data set) number of clusters. Here the parameter $0 < \alpha < 1$ denotes a length-scale that controls the extent of diffusion and F_0 denotes the input to the GROW routine. A propagation step of the form (2) is also used in Pagerank-NIBBLE [5] and NMFR [1], up to replacing WD^{-1} with $D^{-1/2}WD^{-1/2}$ in the latter case. As another example, choosing to sample with or without replacement in the PLANT routine leads to essentially no significant difference in the resultant clusterings.

2.2 A Multigrid Speedup

The main computational burden of the basic INCRES algorithm lies in the GROW routine. Its computational cost scales like $O(R \times E \times \mathrm{diam}(G))$ in the worst case, where E denotes the number of edges in the graph and $\mathrm{diam}(G)$ denotes its diameter. In practice, the *expected* diameter of the graph effectively determines the cost of each step in the algorithm. For graphs commonly used in machine learning, such as k-nearest neighbor graphs, the number of matrix multiplications required in each call to GROW is generally quite small as a result.

However, the computational burden of the GROW routine still causes the straightforward implementation of our algorithm (in serial) to run two orders of magnitude slower than popular multiscale coarsen-and-refine algorithms such as GRACLUS [6] and METIS [7]. As we now show, pursuing a similar coarsen-and-refine strategy allows us remove an additional order of magnitude from the runtime of our algorithm. In many cases, this multiscale version of INCRES still maintains the consistently high level of accuracy obtained by the basic version.

Coarsening Phase: We follow the same coarsening procedure used by GRACLUS and its relatives [6, 7] in our multilevel approach. We construct an agglomerative hierarchy of weighted graphs in a recursive fashion, beginning from the original weighted graph. We therefore set $G^1 := G = (V, W)$ and then successively transform the vertex set $V^1 = V$ into a sequence of smaller weighted graphs $G^2, G^3, \ldots, G^L$ in such a way that the size of each corresponding vertex set decreases $|V^1| > |V^2| > \ldots > |V^L|$ in a geometric fashion. The procedure terminates once the number of vertices $|V^L|$ in the current graph falls below some number n_0, which we take as $n_0 = 20R$ in our experiments.

The transition between two successive levels G^l and G^{l+1} proceeds as follows. Each vertex of G^l begins unmarked. We then visit the vertices in V^l one-by-one according to their degree, from smallest to largest.

When visiting a given vertex v_i, we merge it with its neighboring unmarked vertex v_j that maximizes

$$\frac{W_{ij}}{d_i} + \frac{W_{ij}}{d_j}.$$

The two merged vertices v_i and v_j are then marked, and the process continues. If at any stage a vertex v_i has no unmarked neighbors, we simply mark v_i and leave it as a singleton. The vertices v_i^{l+1} in G^{l+1} therefore correspond to pairs of vertices $\{v_{i_1}^l, v_{i_2}^l\}$ from G^l, so that the number of vertices $|V^l|$ roughly halves at each stage. Given two new vertices $v_i^{l+1} = \{v_{i_1}^l, v_{i_2}^l\}$ and $v_j^{l+1} = \{v_{j_1}^l, v_{j_2}^l\}$, we define the weights $W_{i,j}^{(l+1)}$ between these two vertices according to the relations

$$\begin{aligned} W_{ij}^{(l+1)} &:= W_{i_1 j_1}^{(l)} + W_{i_1 j_2}^{(l)} + W_{i_2 j_1}^{(l)} + W_{i_2 j_2}^{(l)}, \\ W_{ii}^{(l+1)} &:= W_{i_1, i_1}^{(l)} + 2W_{i_1 i_2}^{(l)} + W_{i_2 i_2}^{(l)}, \end{aligned}$$

i.e. simply by summing the weights between all possible pairs of vertices in the two merged pairs.

Base Clustering and Refinement: The clustering phase begins with a base clustering of the coarsest graph G^L in the hierarchy. We simply use the basic version of INCRES applied to G^L to obtain this initial clustering. We then extrapolate this clustering to the next level G^{L-1} in the hierarchy, refine the clustering of G^{L-1} using INCRES again, and repeat until we obtain a clustering of the original graph at the finest level.

The extrapolation procedure we use is straightforward: a clustering of G^{l+1} defines a corresponding clustering of G^l by simply assigning each vertex $v_{i_1}^l$ and $v_{i_2}^l$ in the pair $v_i^{l+1} = \{v_{i_1}^l, v_{i_2}^l\}$ to the cluster of its parent. We subsequently refine this clustering using a slightly modified version the original INCRES procedure. Let $N^l := |V^l|$ denote the number of vertices in the current graph. We set the initial number of seeds to $s_i = .2N^l$, the final number of seeds to $s_f = .5N^l$, and we specify a set number of iterations I^l of INCRES to perform at the current level. We then select the seed increment parameter ds so that the number of seeds s transitions from s_i to s_f in exactly I^l iterations. We perform $I^L = 200$ steps of INCRES at the coarsest level and $I^1 = 1$ step of INCRES at the finest level. We then select the number of iterations I^l to perform at the l^{th} level as a geometrically decreasing progression between these two endpoints.

2.3 Relation with Other Work

Our methodology relies upon and incorporates number of ideas from transductive learning. In particular, we leverage the notion of label propagation [3]. In the standard label propagation framework, an oracle provides a set of labeled points or vertices. These labeled points form either nonzero initial conditions or heat sources for a discrete heat equation on the graph. The second step of the INCRES algorithm (the GROW routine) precisely corresponds with a label propagation of the random labels returned from the first step of the algorithm (the PLANT routine).

The NIBBLE algorithm and its relatives [2, 8, 5, 4] use a similar idea to get an unsupervised clustering method from label propagation by planting random seeds. These works cluster the entire graph in a sequential manner: at each step a single random vertex is drawn and propagated. Then a sweep is performed to extract a small cluster around this vertex. These algorithms function well for problems aimed at extracting many small clusters from graphs with fine structure. In contrast, we perform multiway partitioning directly instead of recursively. We also aim at medium scale clusters instead of small scale clusters. We also utilize a significantly different random seeding strategy.

The INCRES algorithm also alternates between label propagation (GROW) and thresholding (HARVEST). The idea of iteratively alternating between a few steps of label propagation and subsequent thresholding has also appeared in a transductive learning context [9], although the presence of labeled information results in a different implementation of the propagation step. The non-negative matrix factorization method [1] also incorporates random walk information in a manner that resembles the GROW routine, but otherwise the underlying principles of the algorithms differ substantially.

Finally, the algorithms GRACLUS [6] and METIS [7] directly inspired the multigrid version of our algorithm. We use essentially the same coarsening algorithm, but rely upon a different clustering on the coarsest

Table 1: Algorithmic Comparison via Cluster Purity.

Data	**size**	**R**	**RND**	**NCUT**	**LSD**	**NMFR**	**MTV**	**INCRES** *(speed 1)*	**INCRES** *(speed 5)*
20NEWS	20K	20	6.3%	26.6%	34.3%	60.7%	35.8%	**61.0%**	60.7%
CADE	21K	3	15.5%	41.0%	41.3%	52.0%	44.2%	**52.8%**	52.1%
RCV1	9.6K	4	30.3%	38.2%	38.1%	42.7%	42.8%	**54.5%**	51.2%
WEBKB4	4.2K	4	39.1%	39.8%	45.8%	**58.1%**	45.2%	57.1%	56.8%
CITESEER	3.3K	6	21.8%	23.4%	53.4%	**62.6%**	42.6%	62.0%	62.2%
MNIST	70K	10	11.3%	76.9%	75.5%	**97.1%**	95.5%	96.0%	94.0%
PENDIGIT	11K	10	11.6%	80.2%	86.1%	86.8%	86.5%	**88.8%**	85.9%
USPS	9.3K	10	16.7%	71.5%	70.4%	86.4%	85.3%	**87.8%**	87.4%
OPTDIGIT	5.6K	10	12.0%	90.8%	91.0%	**98.0%**	95.2%	97.4%	94.8%

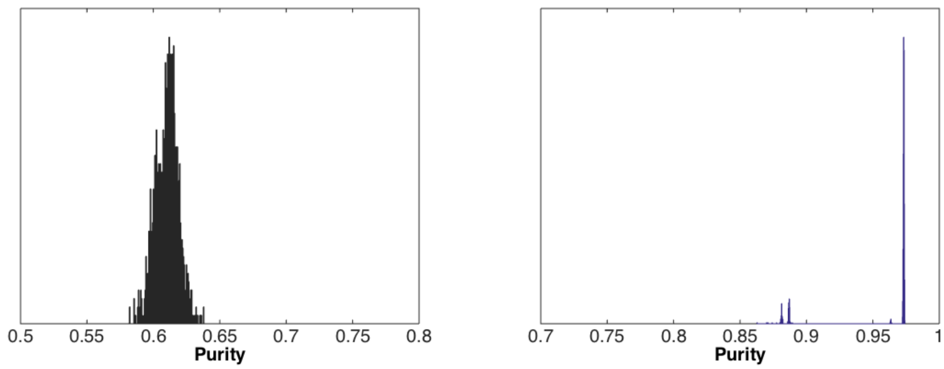
scale (INCRES vs. kernelized k-means or pure spectral clustering). Our refinement technique also differs substantially. The INCRES algorithm relates to the kernelized k-means procedure used in GRACLUS even in the single level case: we can essentially interpret the GROW routine as the "maximization" step in an alternating minimization for a kernelized k-means. However, the kernel is a power of the normalized weights and the power may depend on the cluster, so it is not exactly the same. The "expectation" step in our algorithm is replaced by sampling, and instead of having a single representative for a class, the number of representatives increases as the algorithm progresses. Using power iterations of the weight matrix W directly for clustering has appeared in [10, 11]. These works utilize the power iterations to generate an embedding of the vertices of the graph, which is then clustered using k-means. These methods can also be considered as kernelized k-means methods, with a power of the weights providing the kernel.

Because the GROW function we use iterates the random walk on the graph, our algorithm is a form of spectral clustering. However, our main contribution to the clustering problem, and the primary novelty in our algorithm, is the *incremental reseeding process*. This process is not fundamentally tied to the INCRES algorithm presented here— it seems to be quite universal and can be adapted to other clustering methods. However, combining reseeding with the random walk method offers an excellent combination of accuracy, speed, and robustness.

3 Experiments

We now provide the results of our extensive experimental evaluation of the algorithm. This section shows that our algorithm achieves state-of-the-art performance in terms of cluster purity on a variety of real word data sets while running an order of magnitude faster than the other comparably accurate clustering methods. Moreover, whereas some of the competing algorithms are initialized with a partition provided by a low cost algorithm such as NCut, the INCRES algorithm is initialized with a random partition. Finally we show that INCRES is very robust to perturbations in the input graph.

The Algorithms: We compare our method against four clustering algorithms that rely on variety of different principles. We select algorithms that, like our algorithm, partition the graph in a direct, non-recursive manner. The NCut algorithm [12] is a widely used spectral algorithm that relies on a post-processing of the eigenvectors of the graph Laplacian to optimize the normalized cut energy. The NMFR algorithm [1] uses non-negative matrix factorization and graph-based random walk principles in order to factorize and regularize the original input similarity matrix. The LSD algorithm [13] provides another non-negative matrix factorization algorithm. It aims at finding a left-stochastic decomposition of the similarity matrix. The MTV algorithm from [14] provides a total-variation based algorithm that attempts to find an optimal multiway Cheeger cut of the graph by using ℓ^1 optimization techniques. The last three algorithms (NMFR, LSD and MTV) all use NCut in order to obtain an initial partition. By contrast, we initialize our algorithm with a random partition. We use the



(a) 20NEWS (b) MNIST

Figure 2: Purity Distributions

code available from [12] for NCut, the code available from [1] to test the two non-negative matrix factorization algorithms (NMFR and LSD) and the code available from [14] for the MTV algorithm.

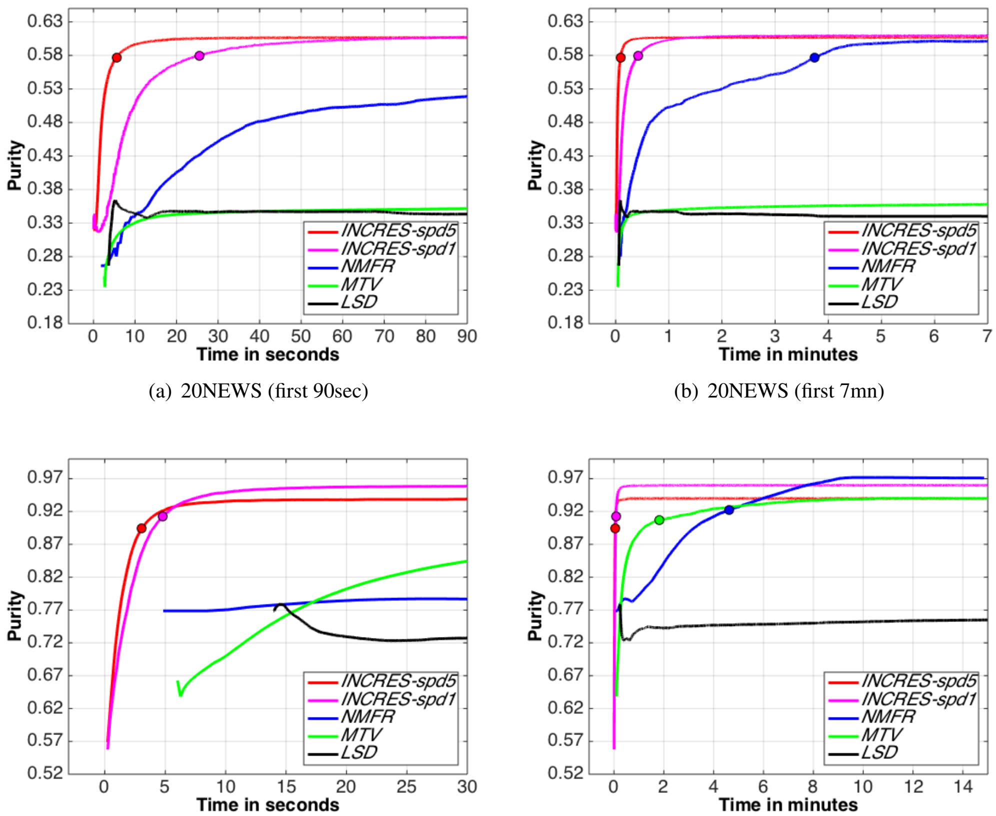
The Data Sets: We provide experimental results on five text data sets (20NEWS, CADE, RCV1, WEBKB4, CITESEER) and four data sets containing images of handwritten digits (MNIST, PENDIGITS, USPS, OPTDIGITS). We processed the text data sets by removing a list of stop words as well as by removing all words with fewer than twenty occurrences (for 20NEWS) and fewer than five occurrences (for all others) across the corpus. We then construct a 5-NN graph based on the cosine similarity between tf-idf features. For variety, we include some weighted graphs (RCV1 and CITESEER) as well as some unweighted graphs (20NEWS, CADE and WEBKB4). For MNIST, PENDIGITS and OPTDIGITS we use the similarity matrices constructed by [1], where the authors first extract scattering features [15] for images before calculating an unweighted 10-NN graph. For USPS we constructed a weighted 10-NN graph from the raw data without any preprocessing. We provide the source for these data sets and more details on their construction in the Appendix.

Accuracy Comparisons: In Table 1 we report the accuracy obtained by the selected algorithms LSD, NMFR, MTV and INCRES (for two values of the timestep parameter, **speed** $= 1$ and **speed** $= 5$) on the various data sets. We use cluster purity to quantify the quality of the calculated partition, defined according to the relation

$$\text{Purity} = \frac{\text{number of “successes”}}{N} = \frac{1}{N}\sum_{r=1}^{R} \max_{1<i<R} n_{r,i}.$$

Here $n_{r,i}$ denotes the number of data points in the r^{th} cluster that belong to the i^{th} ground-truth class. In other words, given a computed cluster we count a data point as a success if it belongs to the ground truth class that best represents the cluster. We allowed each iterative algorithm a total of 10,000 iterations to reach convergence. Both INCRES and MTV rely on randomization, so for these algorithm we report the average purity achieved over 1000 different runs. The fourth column of the table (RND) provides a base-line purity for reference, i.e. the purity obtained by assigning each data point to a class from 1 to R uniformly at random. The boldface numbers in the table indicate the highest purity score achieved on each data set.

Overall, INCRES and NMFR significantly outperform the other algorithms. This is especially true for text data sets. Both algorithms utilize a random walk strategy to help “smooth” irregular graphs, such as the similarity matrices obtained from text data sets. This strategy also contributes to the robustness of these algorithms and to their solid performance across the full range of data sets. However, the INCRES algorithm typically runs at least one order of magnitude faster than the NMFR algorithm. Due to the similarity of their results, we provide a more exhaustive comparison between these two algorithms in the appendix on 17 additional data



(a) 20NEWS (first 90sec)

(b) 20NEWS (first 7mn)

(c) MNIST (first 30sec)

(d) MNIST (first 15mn)

Figure 3: Purity curves for the four algorithms considered on two benchmark data sets (20NEWS and MNIST). We plot purity against time for each algorithm over two different time windows. The circular marks on each curve indicate the point at which the curve reaches 95% of its limiting value. The corresponding times at which this happens are reported in Table 2.

sets.

Finally, note that the INCRES algorithm performs comparably when **speed** $= 1$ and **speed** $= 5$, demonstrating that the algorithm is robust with respect to the choice of the seed increment parameter ds. The INCRES algorithm also performs rather consistently across independent runs of the random process. Figure 2 provides an experimental validation of this fact. Specifically, this figure displays a histogram of the purity values obtained over 1008 independent trials of INCRES on both 20NEWS and MNIST (using **speed** $= 1$ for this example). On 20NEWS, all independent trials of the INCRES algorithm converge to a solution with a purity value between 58% and 64%. The INCRES algorithm converges to one of two attractors on MNIST, one at roughly 88% purity and one at roughly 97% purity, with the vast majority of all trials converging to the higher-accuracy attractor.

Speed Comparisons: Figure 3 illustrates the speed at which the iterative algorithms (LSD, MTV, NMFR and INCRES) converge toward their respective solutions. We ran each algorithm for a total of 7 minutes on 20NEWS and for 15 minutes on MNIST. We report the purity obtained by the algorithm at each iteration. For the randomized algorithm (INCRES and MTV) the purity curves were obtained by averaging the results over 240 runs. The overwhelming computational burden for all of these algorithms arises from the sparse-matrix times full-matrix multiplications required at each step. Each algorithm is implemented in a fair and consistent

Table 2: Computational Time

Data	**NMFR**	**MTV**	**INC.** *(spd1)*	**INC.** *(spd5)*
20NEWS	3.7mn (57.7%)	– –	25.4s (57.7%)	5.6s (58%)
MNIST	4.6mn (92.2%)	1.8mn (90.7%)	4.8s (91.2%)	3.1s (89.3%)

Table 3: Robustness Comparisons

Noise	**NCUT**	**LSD**	**MTV**	**GRACLUS** *(multilevel)*	**INCRES-ml** *(multilevel)*	**INCRES** *(speed1)*
20NEWS						
+0% edges	27%	34%	36%	42%	59%	**61%**
+50% edges	21%	27%	20%	15%	19%	**52%**
+100% edges	18%	22%	11%	13%	12%	**44%**
+150% edges	15%	20%	10%	12%	10%	**34%**
+200% edges	14%	18%	9%	11%	9%	**27%**
MNIST						
+0% edges	77%	76%	96%	**97%**	**97%**	96%
+50% edges	87%	94%	55%	93%	**97%**	**97%**
+100% edges	84%	93%	25%	80%	90%	**97%**
+150% edges	74%	87%	18%	63%	74%	**97%**
+200% edges	67%	82%	16%	52%	53%	**96%**

way, and the experiments were all performed on the same architecture.

In order to give an indication of the speed/accuracy trade-off for each algorithm, in Table 2 we record the time it took for the purity obtained by each algorithm to reach 95% of its limiting value on both 20NEWS and on MNIST. Overall, the simple INCRES algorithm provides accuracy comparable to the state-of-the-art NMF algorithm [1], yet runs an order of magnitude faster. Timing results on the data sets from table 1 are consistent with those obtained for 20NEWS and MNIST, in the sense that INCRES typically runs one order of magnitude faster than NMFR on these data sets as well.

Robustness Experiments: Table 3 reports accuracy results of various algorithms on graphs that we corrupted by adding different levels of noise. We began with the original 20NEWS graph used in Table 1 and added additional edges to the graph uniformly at random. The original graph had $e = 144,632$ edges. For the experiment, we added $0.5e$, e, $1.5e$ and $2e$ additional noise edges. For each of these four levels of noise, we randomly generated 144 separate perturbed graphs. The table reports, for each level of noise, the average purity obtained by each algorithm on the 144 randomly generated matrices. We then proceeded to perturb the original MNIST graph in a similar fashion. The original graph has $e = 1,027,412$ edges, and we randomly generated 120 graphs at each level of noise. This gives a total of 1056 randomly generated graphs for this set of experiments. We provide experimental results for all algorithms other than NMFR, which simply takes far too much time to run to convergence on all 1056 adjacency matrices. Nevertheless, given the similarity in their performace, we expect that NMFR would perform similarly to INCRES on this set of experiments as well.

The results clearly elucidate the robustness of the INCRES algorithm with respect to noise in the graph construction process. On the 20NEWS data set, for example, all other algorithms experience a sharp decrease in accuracy as soon as noise is added. In contrast, the purity of the INCRES algorithm slowly decreases in a stable fashion. On the MNIST data sets, the results obtained by INCRES remain essentially unchanged across all noise levels. The competing algorithms do not exhibit this behavior. Interestingly, NCut and LSD actually obtain *better* results at the 50% and 100% noise levels. Given that LSD relies on NCut for initialization,

Table 4: Accuracy Comparison for Multilevel Algorithms.

Data	**size**	**METIS**	**GRACLUS**		**INCRES-ml**	
20NEWS	20K	42.4%	42.4%	(0.05s)	**58.6%**	(0.85s/0.07s)
CADE	21K	29.3%	**43.5%**	(0.10s)	40.9%	(0.61s/0.29s)
RCV1	9.6K	34.1%	42.4%	(0.01s)	**47.9%**	(0.15s/0.03s)
WEBKB4	4.2K	37.9%	49.0%	(0.01s)	**53.9%**	(0.10s/0.02s)
CITESEER	3.3K	45.2%	53.5%	(0.01s)	**61.5%**	(0.11s/0.02s)
MNIST	70K	86.0%	**96.9%**	(0.17s)	**96.9%**	(1.99s/0.52s)
PENDIGIT	11K	67.3%	84.7%	(0.02s)	**88.8%**	(0.46s/0.05s)
USPS	9.3K	75.1%	86.9%	(0.02s)	**87.2%**	(0.34s/0.05s)
OPTIDIGIT	5.6K	83.0%	94.2%	(0.01s)	**95.0%**	(0.26s/0.03s)

it comes as no surprise that gains for NCut produce subsequent gains for LSD as well. This pathological behavior still indicates a lack of robustness, in the sense that both algorithms exhibit a high degree of sensitivity to changes in the underlying graph.

Multigrid Experiments: Table 4 reports the accuracies and run times of the coarsen and refine algorithms METIS [7], GRACLUS [6] and the multilevel version of our reseeding algorithm. We refer to the multilevel version of INCRES as INCRES-ml to distinguish it from the basic version. The reported purity values correspond to the average accuracy obtained over 500 trials of each routine. By and large, INCRES-ml obtains clustering of higher quality than the two other multilevel algorithms. This comes at a cost of one order of magnitude in computational time.

The computational cost that INCRES-ml incurs at level l of the graph hierarchy scales as

$$O(R \times E_l \times \text{diam}(G_l) \times I^l),$$

where E_l denotes the number of edges in the graph G^l and I^l once again denotes the number of iterations performed at this level. In comparison, the computational burden that GRACLUS incurs at level l scales as

$$O(R \times E_l \times I^l),$$

where I^l has the same meaning as before, but obviously refers to the number of iterations performed by GRACLUS. The extra term diam(G_l) in the computational cost of INCRES-ml partly explains the difference between the computational times. However, the diameters of the graphs considered in our experiments are typically smaller than ten. This is especially true for the coarsest levels in the hierarchy. The other major source of the difference in computational time comes from implementation: while GRACLUS is a heavily optimized C code, our multilevel algorithm has a naive MATLAB implementation. Finally, we remark that all of these multilevel procedures suffer from a severe sensitivity to noise, see Table 3. This fact motivates the use of the basic version of INCRES in situations where robustness to the graph construction process is highly desirable.

4 Appendix

4.1 Matlab Code used in the experimental section

Below is the exact MATLAB code that we used in the experimental section of the paper.

```
function C=INCRES( W , R , speed , maxiter)
D = sum(W,2);
RW=W*spdiags(1./D,0,n,n);
s=1;
ds=speed*10^(-4)*n/R;
C=randi(R,n,1);
for iter=1:maxiter
    [F,R] = PLANT(C, R, round(s));
    while ( min( min(F) ) < eps )
        F= RW * F;
    end
    [¬,C] = max(F,[],2);
    s=s+ds;
end
end
```

```
function [F,R] = PLANT(C,R,s)
n=length(C);
F=zeros(n,R);
EmptyClass=[];
for k=1:R
    idx= find(C==k);
    ClassSize=length(idx);
    if  (ClassSize== 0)
        EmptyClass=[EmptyClass,k];
    else
        idxSeeds = idx( randi(ClassSize,s,1) );
        F(:,k)= full(sparse(idxSeeds,ones(1,s),1,n,1));
    end
end
F(:,EmptyClass)=[];
R=size(F,2);
end
```

4.2 Additional Comparison between INCRES and NMFR

The table above provides a more exhaustive comparison between the INCRES and NMFR algorithms. We selected the twenty largest data sets used in the original NMFR paper [1]. We excluded the ADS data set because the similarity matrix contained negative entries and no algorithm performed better than random on this data set. The similarity matrices were downloaded from `http://users.ics.aalto.fi/rozyang/nmfr/index.shtml`. The similarity matrices for the text data sets 20NEWS, RCV1 and WEBKB4 are different than those presented in the main body of the paper. The authors of the original NMFR paper used the 10,000 words with maximum information gain to construct the similarity matrices associated to these text data sets. We have preferred to simply use the words appearing more than a certain number of times to construct our similarity matrices (in order to avoid using ground truth information in the construction of the similarity matrices). For the NMFR algorithm the results included in the above table are the one reported in [1].

Dataset	size	RAND	NMFR	INCRES (speed 1)
YEAST	1.5K	32%	**55**%	54%
SEMEION	1.6K	13%	**94**%	93%
FAULTS	1.9K	34%	39%	**41**%
SEG	2.3K	16%	**73**%	58%
CORA	2.7K	30%	**47**%	46%
MIREX	3.1K	13%	43%	**45**%
CITESEER	3.3K	21%	44%	**47%**
WEBKB4	4.2K	39%	**63**%	59%
7SECTORS	4.6K	24%	**34**%	32%
SPAM	4.6K	60%	**69**%	**69%**
CURETGREY	5.6K	5%	**28**%	19%
OPTDIGITS	5.6K	12%	**98**%	97%
GISETTE	7.0K	50%	**94**%	**94**%
REUTERS	8.3K	45%	**77**%	76%
RCV1	9.6K	30%	54%	**55**%
PENDIGITS	11K	12%	87%	**89%**
PROTEIN	18K	46%	**50**%	46%
20NEWS	20K	6%	63%	**64**%
MNIST	70K	11%	**97**%	96%
SEISMIC	99K	50%	**59**%	56%

Table 5: Algorithmic comparison via cluster purity.

4.3 Datasets

- 20NEWS (unweighted similarity matrix): The word count from the raw documents was computed using the Rainbow library [16] with a default list of stop words. Words appearing less than 20 times were also removed. The similarity matrix was then obtained by 5 nearest neighbors using cosine similarity between tf-idf features. Source: `http://www.cs.cmu.edu/~mccallum/bow/rainbow/`

- CADE (unweighted similarity matrix): The documents in the Cade12 data set correspond to a subset of web pages extracted from the CAD Web Directory, which points to Brazilian web pages classified by human experts. We only kept the three largest classes. The word count from the raw documents was computed with the Rainbow library [16] as before. Words appearing less than 5 times were removed. The similarity matrix was then obtained by 5 nearest neighbors using cosine similarity between tf-idf features.
  Source: `http://web.ist.utl.pt/%7Eacardoso/datasets/`.

- RCV1 (weighted similarity matrix): This dataset was obtained in preprocessed format from `http://www.cad.zju.edu.cn/home/dengcai/Data/TextData.html` with the tf-idf features were already computed. We then simply used cosine similarity and 5-NN.

- WEBKB4 (unweighted similarity matrix): The word count from the raw documents was done with the Rainbow library [16]. A list of stop word was removed. Words appearing less than 5 times were removed. The similarity matrix was then obtained by 5 nearest neighbors using cosine similarity between tf-idf features. Source: `http://www.cs.cmu.edu/afs/cs/project/theo-20/www/data/`

- CITESEER (weighted similarity matrix): This dataset was obtained in preprocessed format from `http://linqs.cs.umd.edu/projects//projects/lbc/index.html` where each publication in the dataset is described by a 0/1-valued word vector indicating the absence/presence of the corresponding word from the dictionary. We then simply used cosine similarity and 5-NN.

- MNIST, PENDIGITS, OPTDIGITS (unweighted similarity matrix): The similarity matrices were obtained from [1], where the authors first extracted scattering features using [17] for images before calculating the 10-NN graph. Source: `http://users.ics.aalto.fi/rozyang/nmfr/index.shtml`
- USPS (weighted similarity matrix): We computed a 10-NN graph using standard Euclidean distance between the raw images. Each edge in the 10-NN graph was given the weight

  $$w_{ij} = \mathrm{e}^{-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}}$$

  where each $\mathbf{x}_i$ denotes a vector containing the raw pixel data. The parameter σ was chosen as the mean distance between each vertex and its 10^{th} nearest neighbor. Source: `http://www.cad.zju.edu.cn/home/dengcai/Data/MLData.html`

References

[1] Zhirong Yang, Tele Hao, Onur Dikmen, Xi Chen, and Erkki Oja. Clustering by nonnegative matrix factorization using graph random walk. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1088–1096, 2012.

[2] László Lovász and Miklós Simonovits. Random walks in a convex body and an improved volume algorithm. *Random structures & algorithms*, 4(4):359–412, 1993.

[3] Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. Semi-supervised learning using gaussian fields and harmonic functions. In *IN ICML*, pages 912–919, 2003.

[4] Daniel A. Spielman and Shang-Hua Teng. A local clustering algorithm for massive graphs and its application to nearly linear time graph partitioning. *SIAM Journal on Computing*, 42(1):1–26, 2013.

[5] Reid Andersen, Fan Chung, and Kevin Lang. Local graph partitioning using pagerank vectors. In *Proceedings of the 47th Annual Symposium on Foundations of Computer Science (FOCS '06)*, pages 475–486, 2006.

[6] Inderjit S. Dhillon, Yuqiang Guan, and Brian Kulis. Weighted graph cuts without eigenvectors: A multilevel approach. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(11):1944–1957, 2007.

[7] George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput.*, 20(1):359–392, 1998.

[8] Daniel A. Spielman and Shang-Hua Teng. Nearly-linear time algorithms for graph partitioning, graph sparsification, and solving linear systems. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 81–90, 2004.

[9] Cristina Garcia-Cardona, Ekaterina Merkurjev, Andrea L. Bertozzi, Arjuna Flenner, and Allon G. Percus. Multiclass data segmentation using diffuse interface methods on graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 99:1, 2014.

[10] Frank Lin and William W. Cohen. Power iteration clustering. In *ICML*, pages 655–662, 2010.

[11] Stéphane Lafon and Ann B. Lee. Diffusion maps and coarse-graining: A unified framework for dimensionality reduction, graph partitioning, and data set parameterization. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(9):1393–1403, 2006.

[12] Stella X. Yu and Jianbo Shi. Multiclass spectral clustering. in international conference on computer vision. In *International Conference on Computer Vision*, 2003.

[13] Raman Arora, M Gupta, Amol Kapila, and Maryam Fazel. Clustering by left-stochastic matrix factorization. In *International Conference on Machine Learning (ICML)*, pages 761–768, 2011.

[14] Xavier Bresson, Thomas Laurent, David Uminsky, and James von Brecht. Multiclass total variation clustering. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

[15] Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1872–1886, 2013.

[16] Andrew Kachites McCallum. Bow: A toolkit for statistical language modeling, text retrieval, classification and clustering. http://www.cs.cmu.edu/ mccallum/bow, 1996.

[17] M. Stephane. Group invariant scattering. *Communications on Pure and Applied Mathematics*, 65(10):1331–1398, 2012.