Large-scale Image Labeling via Mapreduce Topic Modeling

[Spring 2012 CMSC828G Final Project Report] *

Khoa Doan
University of Maryland
Dept. of Computer Science
College Park, MD 20742
trovato@corporation.com

Rahmatri Mardiko
University of Maryland
Dept. of Computer Science
College Park, MD 20742
mardiko@cs.umd.edu

Ang Li
University of Maryland
Dept. of Computer Science
College Park, MD
angli@cs.umd.edu

ABSTRACT

Large scale computer vision is recently being popular due to the emerging of large scale imagery data and in need of more generally trained visual models. Recent advances in feature extraction provide a feasible and succinct way to represent image regions. However, most of these features are computationally heavy to extract. Since there is no explicit method to essentially speed up feature extraction, parallelization becomes the most comfortable strategy for this task. Due to the inherent high dimensionality of visual data, extracted features can be very noisy and thus not representative for images. Feature quantization is introduced to group similar features into the same low level semantics. K-means clustering is one of the usual choices for quantization while it may suffer from either time or space problems in ordinary environment. Latent Dirichlet Allocation is one of the techniques to discover higher level topic semantics for the context, which was popular in natural language processing. In this project, we unearth the potential of adapting computer vision tasks such as low level feature extraction and higher level image understanding into the MapReduce framework for large scale image dataset. We adopt the MapReduce LDA (Mr.LDA) to find topics across the images and discuss its possible extensions with respect to the image domain.

Keywords

Topic modeling, image labeling, parallelization, MapReduce framework

1. INTRODUCTION

Large scale visual data analysis and machine learning have been recently received much attention during the past few years. Traditional computer vision research focuses on small datasets of images or videos which makes the generalization of these methods difficult. In the current world of big data, a lot of imagery data have been present in the Internet. How to make use of the large scale of data to explore a better visual model has been one of the central problems in the current community. However, one natural problem arised from this task is heavy computational load. Computer vision tasks are usually involved with feature engineering which requires a lot of time and space for experiments. Fortunately, the MapReduce framework provides a reliable approach to large scale data processing.

In this work, we explore the potentials of using MapReduce framework for a standard computer vision task i.e. image labeling. The objective of the image labeling task is to find underlying semantics for each of the image pixels and to find groups of pixels that belong to the same object or semantic. Image labeling has been investigated for decades, although the scale of the dataset is limited due to computational issues. However, the demand of large scale image labeling turns out to be more and more clear in the recent years. One of the reasons is that people are being aware of using computer techniques to assist human annotation for specific imagery data such as remote sensing data. Millions of satellite images are generated for every day and the task of understanding these data is never feasible for human to do exhaustively. The automatic way will benefit the community tremendously in different areas such as surveillance, city planning, national defense, etc.

The rest of this paper is organized as follows. Section 2 briefly introduces the system structure of this project. MapReduce framework for visual feature extraction is discussed in section 3. Details of Mr.LDA is introduced in section 4. In section 6, a few implementation details are discussed. Section 7 shows the evaluation methods and experimental results for this project. Discussion on possible extensions from LDA to Spatial LDA is presented in section 8. Finally, the paper concludes in section 9.

2. SYSTEM OVERVIEW

3. FEATURE EXTRACTION

Feature extraction often dominates the most computational resources in computer vision tasks. In this section, we introduce a MapReduce based framework to extract visual features efficiently which distributes the computation loads for raw feature extraction and scales up the construction of feature codebook via MapReduce K-means clustering.

3.1 Scale Invariant Feature Transform (SIFT)

In this work, the Scale Invariant Feature Transform (SIFT) [4] is adopted to describe local regions across the images.

*Data-Intensive Computing with MapReduce

Generally, a SIFT descriptor represent a region bounded by a rectangular box centered at (x_c, y_c). The bounding box is uniformly divided into $K \times K$ parts. The gradients are computed in these sub-regions and the gradient magnitudes in N_b (the number of bins) discretized orientations are computed. A histogram is then constructed to represent the distribution of the gradient magnitudes. Therefore, for each region, the SIFT descriptor is a $K^2 N_b$ dimensional histogram vector.

One of the advantages of SIFT descriptors is its invariance to image rotation and translation, which makes this feature very popular among the computer vision community. We base our project on the MPI-CBG JavaSIFT library[2].

3.2 Dense Sampling SIFT Features

In order to apply SIFT features to represent the whole image for the task of image labeling, each of the pixels ideally should be described. Due to the fact that a neighborhood of pixels usually have little difference and belong to the same semantics, we sample series of small rectangular regions with a parameter STEPSIZE densely across each of the image and compute the SIFT description for each of the regions. The STEPSIZE controls the number of pixels between two consecutive centers of bounding boxes. Thereafter, each image can be converted from visual data to a list of (KEY,VALUE) pairs where the KEY is the index (g_i, x_i, y_i) of the regions and the VALUE is the SIFT feature vector f_i. g_i is the id of the image that contains the i-th region and (x_i, y_i) is the center location of the i-th region.

We need a graph to illustrate the feature extraction strategy.

Figure 1: Illustration of Dense SIFT feature extraction

3.3 Image Input Aggregation

Given a large number of image files as input, there are several ways of loading them into a MapReduce job. One possible way is creating a text file containing a list of file names and let the mapper reads the files from HDFS and processes them while the reducer does nothing. An alternative way is similar to the first except the reading and processing is moved to the reducer while the mapper just passes the (key,value) pairs. Both are relatively simple and easy to implement. However, due to the small-files problem in Hadoop[5], neither the first and the second approach achieves the best performance in terms of processing speed.

Since Hadoop framework works best with large files, we need to aggregate the images in the dataset into a few big files. This step allows us to speed-up the feature extraction (and other images processing tasks) given a large number of image files as input. We adopt the approach presented in[6] which creates a MapReduce job that packs multiple files into a `SequenceFile`. This aggregation is performed as a preprocessing step for the other jobs that take images as input.

3.4 MapReduce-based Feature Extraction

A list of image paths is generated in order to locate the image files. Each of the paths is assigned a number as image ID. The MapReduce job takes the path list as input and output sequence files containing region indices as keys and SIFT feature descriptors as values.

Algorithm 1: Mapreduce based Visual Feature Extraction

```
1  class Mapper
2      method map(LongWritable key, Text value)
3          (id, path) = split(value);
4          emit (id, path)
5  class Reducer
6      method reduce(IntWritable key, Text value)
7          id = key;
8          image = readImageFromHDFS(value);
9          feature = extractFeature(image);
10         emit (id, feature)
```

3.5 Building Feature Codebook

The raw features extracted from above generally have two problems in representing the images. On the one hand, the dimension of each feature vector is typically 128. For the similarity measure of each pair of small regions, at least 128 times of multiplications are necessary for Euclidean distances. This makes the further processing intractable even using parallelization framework. On the other hand, the feature vector is a histogram of oriented gradients which contains a lot of noises. Therefore, feature quantization is introduced to further processing the features and construct a "codebook" for the features. In the codebook, nearby feature vectors are grouped into the same index because they should belong to the same visual semantics.

K-means clustering is usually the choice for doing feature quantization. However, the clustering takes more features and thus needs much more memory spaces in order to exhaustively compute the distances between cluster centers to each of the points, as the scale of data becomes larger. The MapReduce framework generally resolve this time and space problem because not only the computation is distributed into different nodes but also the intermediate results of distances are stored almost uniformly among all of the nodes. In our project, we adopt Mahout K-means clustering [1] for building the codebook in Hadoop environment.

4. MAPREDUCE LDA

[7]

5. MAPREDUCE SPATIAL LDA

6. IMPLEMENTATION REMARKS

In this section we present some implementation details that we did in the project.

6.1 Image-to-SequenceFile Conversion

In the evaluation phase of the project we need to compare the LDA output and the true labels. To enable pixel-by-pixel evaluation we extract each pixel in the ground truth images and store them as (key,value) pairs. The key, which contains the image id and the pixel position (x,y), is of type `TripleOfInts` whereas the value is of type `IntWritable` and

it contains the pixel value. This images-to-pixels conversion can actually be performed as map-only MapReduce job. However, sometimes it is useful to group pixels that have the same semantic together. Here we can apply value-to-key conversion so the pixels that have the same labels are grouped together when they arrive at the reducer.

Algorithm 2: Conversion from Images to SequenceFiles

```
1  class Mapper
2      method map(IntWritable key, BytesWritable value)
3          image = readImageFromBytes(value);
4          (width, height) = sizeOf(image);
5          for y = 1 → height do
6              for x = 1 → width do
7                  pixel = getPixel(image, x, y);
8                  emit (pixel, triple(key, x, y));
9  class Reducer
10     method reduce(IntWritable key,
       Iterable<TripleOfInts> value)
11         iter = values.iterator();
12         while iter.hasNext() do
13             emit (iter.next(), key);
```

6.2 Sequencefile-to-Image Conversion

Also for the evaluation purpose, we need to build images from a set of pixels. This task is the inverse of the previous task. The mapper takes ((image_id,x,y),pixel) pairs and produce (image_id,(x,y,pixel)) as intermediate key value pairs so the pixels that belong to the same image are grouped together in the reducer. In the reducer the pixels are used to build the image object and convert it to `BytesWritable`. Since the output is in the form of `SequenceFile`, we still need to read the output separately to get the individual image files.

Algorithm 3: Conversion from SequenceFiles to Images

```
1  class Mapper
2      method map(TripleOfInts key, IntWritable value)
3          (id, x, y) = getMembers(key);
4          emit (id, triple(x, y, value));
5  class Reducer
6      method reduce(IntWritable key,
       Iterable<TripleOfInts> value)
7          image = new image(width,height);
8          iter = values.iterator();
9          while iter.hasNext() do
10             (x, y, pixel) = getMembers(value);
11             setPixel(image, x, y, pixel);
12         bytes = getBytes(image);
13         emit (key, bytes);
```

7. EXPERIMENTAL EVALUATION

7.1 Dataset

7.1.1 MSRC Image Labeling Dataset

The MSRC image labeling dataset (version 1) [3] contains 240 images and 9 object classes in total. Each of the images is pixel-wise labeled. Fig.(2) shows some sample images from MSRC dataset.

Figure 2: Sample images from MSRC Image Labeling Dataset

7.1.2 Eastern Coast Satellite Image Dataset

We also collect satellite images using Google Maps APIs along the eastern coast in the United States. Each of the image is of 8-bit colormap PNG format and contains 402 × 415 pixels. Due to the capacity of our cluster, we pick 10,000 images amongst the dataset for the experiments. Fig.(3) shows some sample images from Satellite image dataset.

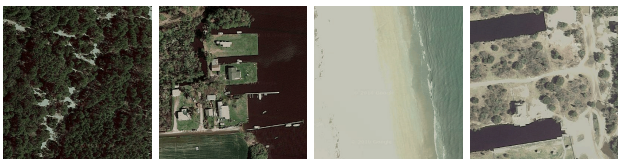
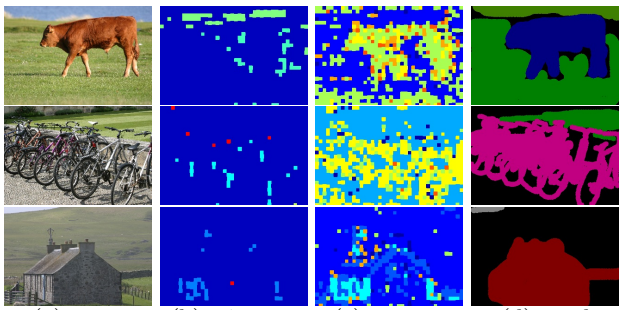


Figure 3: Sample images from Eastern Coast Satellite Image Dataset

7.2 Qualitative Evaluation

Images with labels in different colors are recovered from the output of LDA to show qualitatively which parts belong to the same semantics. Figure 4 presents three sample images and the output labels with 9 and 14 topics. The ground truth consists of 14 distinct semantics. However, the performance of LDA is significantly higher with 9 topics than 14 topics.



(a) origin (b) 14 topics (c) 9 topics (d) truth

Figure 4: Sample LDA output from MSRC dataset. The second and the third column show the output of labeling with LDA using 14 and 9 topics. The last column show the ground truth images.

7.3 Quantitative Evaluation

Traditional computer vision dataset has ground-truth for evaluation. For the MSRC image labeling data, each image

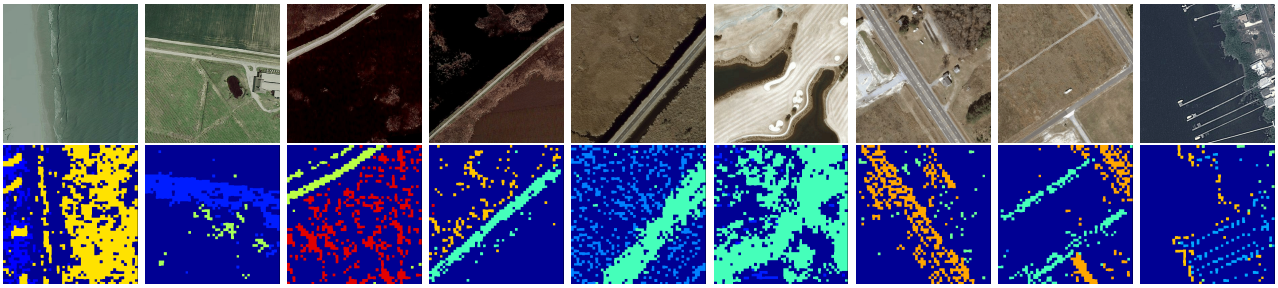


Figure 5: Qualitative results for Google Maps Satellite Image Dataset

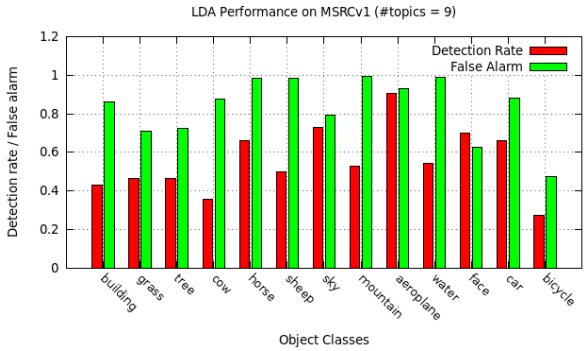
is corresponded to a groundtruth image in which the whole image is divided into several components. Each component is annotated using a unique RGB color. In our output, each pixel is assigned to a type of semantics. Since our approach is completely unsupervised, the truth meaning of each semantic is not understandable. One possible way to evaluate the performance is to enumerate every possible correspondence between the two set of semantics. Apparently this strategy only works when the number of topics is small enough. It becomes difficult to construct a one-one correspondence between our labels and the groundtruth labels. In fact, how to accurately evaluate the performance of image segmentation tasks is still an open problem.

In this project, we propose to evaluate the results for each of the topics separately. For each semantic in the ground truth, a binary image can be constructed, regions with value 1 are those containing such semantic and otherwise not. For the output of the resulted topics, the topic that has the most number of occurrences in the semantic regions are chosen as a correspondence. The precision and recall are then computed according to that topic. Suppose the ground truth is $G : \mathbf{x}(id, x, y) \rightarrow \{1, 2, \ldots, K\}$ and the labeling map is $P : \mathbf{x}(id, x, y) \rightarrow \{1, 2, \ldots, K'\}$, then the evaluation score for each class k can be computed as

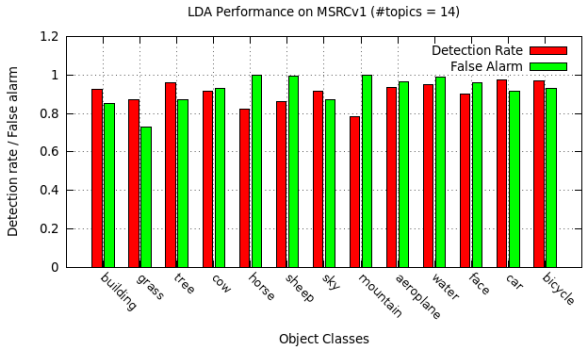
$$\text{DetectionRate}(k) = \frac{\text{cardinal}\left(\{\mathbf{x} \mid P(\mathbf{x}) = k\}\right)}{\text{cardinal}(\{\mathbf{x} \mid G(\mathbf{x}) = k\})} \quad (1)$$

$$\text{FalseAlarm}(k) = \frac{\text{cardinal}(\{\mathbf{x} \mid G(\mathbf{x}) \neq k\})}{\text{cardinal}(\{\mathbf{x} \mid P(\mathbf{x}) = k\})} \quad (2)$$

where cardinal$(\cdot)$ denotes the size of the set.



(a) 9 topics



(b) 14 topics

Figure 6: Quantitative evaluation for each classes

8. DISCUSSION: SPATIAL LATENT DIRICHLET ALLOCATION

9. CONCLUSION

Acknowledgments

The authors would like to thank Prof. Jordan Boyd-Graber and Prof. Jimmy Lin for their consistent help in this project.

10. REFERENCES

[1] Mahout: http://mahout.apache.org/.

[2] mpi-cbg javasift library: http://fly.mpi-cbg.de/ saalfeld/projects/javasift.html.

[3] Msrc dataset: http://research.microsoft.com/en-us/projects/objectclassrecognition/.

[4] D. G. Lowe. Distinctive image features from Scale-Invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004.

[5] T. White. The small files problem: http://blog.cloudera.com/blog/2009/02/the-small-files-problem/.

[6] T. White. *Hadoop: The Definitive Guide*, pages 239–245. O'Reilly Media, Inc., 1st edition, 2009.

[7] K. Zhai, J. B. Graber, N. Asadi, and M. L. Alkhouja. Mr. LDA: a flexible large scale topic modeling package using variational inference in MapReduce. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 879–888, New York, NY, USA, 2012. ACM.

APPENDIX

.0.1 *Display Equations*

A numbered display equation – one set off by vertical space from the text and centered horizontally – is produced by the **equation** environment. An unnumbered display equation is produced by the **displaymath** environment.

Again, in either environment, you can use any of the symbols and structures available in LaTeX; this section will just give a couple of examples of display equations in context. First, consider the equation, shown as an inline equation above:

$$\lim_{n \to \infty} x = 0 \tag{3}$$

Notice how it is formatted somewhat differently in the **displaymath** environment. Now, we'll enter an unnumbered equation:

$$\sum_{i=0}^{\infty} x + 1$$

and follow it with another numbered equation:

$$\sum_{i=0}^{\infty} x_i = \int_{0}^{\pi+2} f \tag{4}$$

just to demonstrate LaTeX's able handling of numbering.

.1 Citations

Citations to articles [**?**, **?**, **?**, **?**], conference proceedings [**?**] or books [**?**, **?**] listed in the Bibliography section of your article will occur throughout the text of your article. You should use BibTeX to automatically produce this bibliography; you simply need to insert one of several citation commands with a key of the item cited in the proper location in the `.tex` file [**?**]. The key is a short reference you invent to uniquely identify each work; in this sample document, the key is the first author's surname and a word from the title. This identifying key is included with each item in the `.bib` file for your article.

The details of the construction of the `.bib` file are beyond the scope of this sample document, but more information can be found in the *Author's Guide*, and exhaustive details in the *LaTeX User's Guide*[**?**].

This article shows only the plainest form of the citation command, using `\cite`. This is what is stipulated in the SIGS style specifications. No other citation format is endorsed.

.2 Tables

Because tables cannot be split across pages, the best placement for them is typically the top of the page nearest their initial cite. To ensure this proper "floating" placement of tables, use the environment **table** to enclose the table's contents and the table caption. The contents of the table itself must go in the **tabular** environment, to be aligned properly in rows and columns, with the desired horizontal and vertical rules. Again, detailed instructions on **tabular** material is found in the *LaTeX User's Guide*.

Immediately following this sentence is the point at which Table 1 is included in the input file; compare the placement of the table here with the table in the printed dvi output of this document.

Table 1: Frequency of Special Characters

Non-English or Math	Frequency	Comments
Ø	1 in 1,000	For Swedish names
π	1 in 5	Common in math
\$	4 in 5	Used in business
Ψ_1^2	1 in 40,000	Unexplained usage

Figure 7: A sample black and white graphic (.eps format).

To set a wider table, which takes up the whole width of the page's live area, use the environment **table*** to enclose the table's contents and the table caption. As with a single-column table, this wide table will "float" to a location deemed more desirable. Immediately following this sentence is the point at which Table 2 is included in the input file; again, it is instructive to compare the placement of the table here with the table in the printed dvi output of this document.

.3 Figures

Like tables, figures cannot be split across pages; the best placement for them is typically the top or the bottom of the page nearest their initial cite. To ensure this proper "floating" placement of figures, use the environment **figure** to enclose the figure and its caption.

This sample document contains examples of **.eps** and **.ps** files to be displayable with LaTeX. More details on each of these is found in the *Author's Guide*.

As was the case with tables, you may want a figure that spans two columns. To do this, and still to ensure proper "floating" placement of tables, use the environment **figure*** to enclose the figure and its caption.

Note that either **.ps** or **.eps** formats are used; use the `\epsfig` or `\psfig` commands as appropriate for the different file types.

.4 Theorem-like Constructs

Other common constructs that may occur in your article are the forms for logical constructs like theorems, axioms, corollaries and proofs. There are two forms, one produced by the command `\newtheorem` and the other by the command `\newdef`; perhaps the clearest and easiest way to distinguish them is to compare the two in the output of this sample document:

This uses the **theorem** environment, created by the `\newtheorem` command:

THEOREM 1. *Let f be continuous on $[a, b]$. If G is an*

Figure 8: A sample black and white graphic (.eps format) that has been resized with the `epsfig` command.

Table 2: Some Typical Commands

Command	A Number	Comments
`\alignauthor`	100	Author alignment
`\numberofauthors`	200	Author enumeration
`\table`	300	For tables
`\table*`	400	For wider tables

Figure 9: A sample black and white graphic (.ps format) that has been resized with the `psfig` command.

antiderivative for f on $[a, b]$, then

$$\int_a^b f(t)dt = G(b) - G(a).$$

The other uses the **definition** environment, created by the `\newdef` command:

Definition 1. If z is irrational, then by e^z we mean the unique number which has logarithm z:

$$\log e^z = z$$

Two lists of constructs that use one of these forms is given in the *Author's Guidelines*.

and don't forget to end the environment with figure*, not figure!

There is one other similar construct environment, which is already set up for you; i.e. you must *not* use a `\newdef` command to create it: the **proof** environment. Here is a example of its use:

PROOF. Suppose on the contrary there exists a real number L such that

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = L.$$

Then

$$l = \lim_{x\to c} f(x) = \lim_{x\to c} \left[gx \cdot \frac{f(x)}{g(x)}\right] = \lim_{x\to c} g(x) \cdot \lim_{x\to c} \frac{f(x)}{g(x)} = 0 \cdot L = 0,$$

which contradicts our assumption that $l \neq 0$. □

Complete rules about using these environments and using the two different creation commands are in the *Author's Guide*; please consult it for more detailed instructions. If you need to use another construct, not listed therein, which you want to have the same formatting as the Theorem or the Definition[**?**] shown above, use the `\newtheorem` or the `\newdef` command, respectively, to create it.

A *Caveat* for the TEX Expert

Because you have just been given permission to use the `\newdef` command to create a new form, you might think you can use TEX's `\def` to create a new command: *Please refrain from doing this!* Remember that your LATEX source code is primarily intended to create camera-ready copy, but may be converted to other forms – e.g. HTML. If you inadvertently omit some or all of the `\def`s recompilation will be, to say the least, problematic.

A. CONCLUSIONS

This paragraph will end the body of this sample document. Remember that you might still have Acknowledgments or Appendices; brief samples of these follow. There is still the Bibliography to deal with; and we will make a disclaimer about that here: with the exception of the reference to the LATEX book, the citations in this paper are to articles which have nothing to do with the present subject and are used as examples only.

B. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you. In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the **.cls** and **.tex** files that it describes.

APPENDIX

A. HEADINGS IN APPENDICES

The rules about hierarchical headings discussed above for the body of the article are different in the appendices. In the **appendix** environment, the command **section** is used to indicate the start of each Appendix, with alphabetic order designation (i.e. the first is A, the second B, etc.) and a title (if you include one). So, if you need hierarchical structure *within* an Appendix, start with **subsection** as the highest level. Here is an outline of the body of this document in Appendix-appropriate form:

A.1 Introduction

A.2 The Body of the Paper

A.2.1 Type Changes and Special Characters

A.2.2 Math Equations

Inline (In-text) Equations

Display Equations

A.2.3 Citations

A.2.4 Tables

A.2.5 Figures

A.2.6 Theorem-like Constructs

A Caveat for the TEX Expert

A.3 Conclusions

Figure 10: A sample black and white graphic (.eps format) that needs to span two columns of text.

A.4 Acknowledgments

A.5 Additional Authors

This section is inserted by LaTeX; you do not insert it. You just add the names and information in the `\additionalauthors` command at the start of the document.

A.6 References

Generated by bibtex from your .bib file. Run latex, then bibtex, then latex twice (to resolve references) to create the .bbl file. Insert that .bbl file into the .tex source file and comment out the command `\thebibliography`.

B. MORE HELP FOR THE HARDY

The acm_proc_article-sp document class file itself is chockfull of succinct and helpful comments. If you consider yourself a moderately experienced to expert user of LaTeX, you may find reading it useful but please remember not to change it.