

Image Compression in Real-Time Multiprocessor Systems Using Divisive K-Means Clustering

Dmitriy Fradkin
Dept. of Computer Science
dfradkin@paul.rutgers.edu
(732)-445-4578

Rutgers, The State University of New Jersey
110 Frelinghuysen Rd., Piscataway, NJ 08854-8019

Ilya B. Muchnik
DIMACS
muchnik@dimacs.rutgers.edu
(732)-445-0073

Simon Streltsov
Mercury Computer Systems, Inc
sstrelts@mc.com
(978)-256-1300
199 Riverneck Rd.
Chelmsford, MA 01824-2820

Abstract—*In recent years, clustering became one of the fundamental methods of large dataset analysis. In particular, clustering is an important component of real-time image compression and exploitation algorithms, such as vector quantization, segmentation of SAR, EO/IR, and hyperspectral imagery, group tracking, and behavior pattern analysis. Thus, development of fast scalable real-time clustering algorithms is important to enable exploitation of imagery coming from surveillance and reconnaissance airborne platforms. Clustering methods are widely used in pattern recognition, data compression, data mining, but the problem of using them in real-time systems has not been a focus of most algorithm designers. In this paper, we describe a practical clustering procedure that is designed specifically for compression of 2-D images and can satisfy stringent requirements of real-time onboard processing.*

1. INTRODUCTION

Many applications can benefit from high compression of images even with coarse approximation. This is particularly important for real-time applications where transfer of information is expensive and time-consuming. Multi-stage high-quality methods such as JPEG [1] may be too computationally expensive, inefficient, and inflexible in such circumstances. This makes it necessary to develop specialized algorithms for fast coarse compression. In this paper, we propose and validate such a method, designed specifically for compression of two-dimensional images. The method is based on a divisive hierarchical clustering procedure applied directly to the pixel space. This method is able to provide fast coarse approximations and is well-suited for implementation on embedded vector processors, such as PowerPC AltiVec.

This paper is organized as follows. We first describe compression requirements of current Department of Defense (DoD) airborne surveillance systems (Section 2). We then describe our method of compression/decompression (Section 3) and discuss its computational requirements in Section 4. Sections 5 and 6 describe the data and provide experimental comparison of the proposed method with standard JPEG compression algorithm. In Section 7, we discuss results and promising directions for future work.

2. MILITARY REQUIREMENTS FOR COMPRESSION

Current fleet of surveillance aircraft, primarily Unmanned Aerial Vehicles (UAVs) Predator and Global Hawk, produce so much data that there is not enough satellite bandwidth to send this data back to the ground. For example, during Kosovo operations in 1999, it was possible to find only 1 gigabit per second (gbps) of bandwidth [2]. At the same time, one Global Hawk UAV consumes 0.5 gbps. As a result, the number of surveillance platforms that can be used during major operations is severely limited by the availability of the bandwidth. The near future is not looking much better: Defense Department plans to launch its own satellites that will amount up to additional 7.7 gbps in 2006. At the same time, DoD UAV roadmap assumes that in 2005 all video surveillance will switch to HDTV format that requires 1 gbps per platform [3]. A similar severe bandwidth limitation when transmitting imagery is present onboard of satellite platforms used by DoD and NASA.

One approach to overcome bandwidth limitation is to process data using registration, classification, tracking algorithms onboard of the UAVs and download only the output of these algorithms. This approach presents severe size and power limitations on processing algorithms, but is preferred when feasible. [4] discusses implementation of this approach on embedded multiprocessors produced by Mercury Computer Systems. When onboard data exploitation is not feasible, system designers have to resort to onboard compression

of imagery data before transmitting it to the ground. This paper assumes implementation of compression algorithm on general-purpose scalable commercial off-the-shelf (COTS) multicomputers, such as PPC-based systems produced by Mercury [5]. Given that these computers are already present on Predator, Global Hawk, and other similar platforms, it will be feasible to add advanced compression algorithms to existing systems.

Compressing imagery onboard of UAVs makes following requirements of a compression algorithm:

- High compression ratio: typical compression ratios can reach 10:1 and 100:1.
- Fast encoding performance on embedded computers: when encoding is performed onboard of UAVs in real time, minimizing size, weight, and power dedicated to compression is very important.
- Scalability in algorithm performance and required computing resources: due to changing mission requirements and other onboard computations that may compete for resources, algorithm should be able to trade between compression ratio, image distortion and computing resources.

3. ALGORITHM DESCRIPTION

Our approach to image compression is to view this problem as a specific case of constructing an approximation to a function defined on the grid. We will refer to the smallest unit of the grid as a pixel. An additional constraint is that all computations are done in integer arithmetic.

K-Means algorithms are popular and widely applied clustering methods. They partition the data to minimize the criterion:

$$E = \sum_{j=1}^K \sum_{x_i \in S_j} d^2(x_i, s_j) \quad (1)$$

where K is the number of clusters, $s_j = \frac{\sum_{x_i \in S_j} x_i}{|S_j|}$ (i.e. s_j is the center of cluster S_j), $d(x, y)$ is Euclidean distance. This criterion was first described by Lloyd [6], Fisher [7] and MacQueen [8]. The above criterion is very general and is usually applied to data in vector form.

K-Means clustering algorithms usually involve choosing a random initial partition or centers, and repeatedly recomputing the centers based on partition and then recomputing the partition based on the centers. Such procedure can be proven to converge to a local minimum, while the problem of finding the global minimum is NP-hard.

We propose a hierarchical divisive K-Means algorithm that minimizes the same criterion (1) as the standard K-Means

with clustering procedure organized as a hierarchical process. This procedure is structurally similar to combinatorial wavelet clustering suggested by B. Mirkin [9]. Such realization of K-Means relies to a large extent on the grid structure of images and cannot be applied to generic vector data.

For the case of grid data, we can re-write criterion (1) as follows:

$$E = \sum_{j=1}^K \sum_{(x_i, y_i) \in A_j} (f(x_i, y_i) - F(A_j))^2 \quad (2)$$

where K is the number of regions, $F(A_j)$ is the mean value of pixels in rectangle A_j :

$$F(A) = \frac{\sum_{(x, y) \in A} p(x, y)}{|A|} \quad (3)$$

and the size of A ($|A|$) is the number of pixels in A .

At each step, our method partitions the grid (or its part) into two rectangles. The partition is chosen among all horizontal and vertical partitions to minimize criterion (2). Such partitioning splits the image into rectangles in which pixel values are very similar to each other. At the end, we choose a representative value $r(A)$ for the region instead of remembering the value of each pixel.

A natural choice for a representative value of pixels in region A is the average pixel value in A , $F(A)$ (as defined in (4)):

$$r_1(A) = F(A) \quad (4)$$

This choice is not unique. For example, in some situations it might be advantageous to use values only from a specific set S (if there are few distinct pixel values in the image) or if there are outside constraints. Then we can use the value in S that is the closest to $F(A)$ as the representative value $r(A)$. Thus

$$r_2(A) = \operatorname{argmin}_{v \in S} |F(A) - v| \quad (5)$$

Yet another possibility is to choose the most frequent value in the region as the representative.

The choice of the representative value function r is important since it affects when we should stop partitioning the grid. Intuitively, we should stop when the pixels are similar to each other and to the chosen representative value, or, in other words, when “diversity” (or approximation accuracy) of a region, $q(A)$, is smaller than some pre-defined threshold. We experimented with two possible approximation criteria:

$$q_1(A) = \max_{(x_i, y_i) \in A} |f(x_i, y_i) - r(A)| \quad (6)$$

$$q_2(A) = \frac{\sum_{(x_i, y_i) \in A} |f(x_i, y_i) - r(A)|}{|A|} \quad (7)$$

The first of these criteria reflects maximal difference between the pixel values in the region and the representative value, while the second one measures the average difference between the pixel values and representative value. Other possible choices for $q(A)$ include:

$$q_3(A) = \frac{\sum_{(x_i, y_i) \in A} (f(x_i, y_i) - r(A))}{|A|} \quad (8)$$

$$q_4(A) = \frac{1}{|A|} \sum_{(x_i, y_i) \in A} \frac{(f(x_i, y_i) - r(A))}{r(A)} \quad (9)$$

The choice of $q(A)$ strongly affects performance and quality of the results. We note that our algorithm automatically determines the number of clusters K (based on function $r(A)$ and $q(A)$ chosen and on the value of ϵ).

Algorithm Structure

We assume that we are given a 2-dimensional grid G of pixel values $f(x, y)$ (where x and y are coordinates), and that the grid size along X and Y axis are given. We are also given an error level ϵ ($0 \leq \epsilon \leq 1$) that controls quality of approximation.

Preprocessing Step: We find the average pixel value of the whole grid F_G and define the threshold $\tau = \epsilon * F(G)$. We will use τ to determine whether the approximation error in a particular region is below the required level.

Let us describe a rectangle A (a part of the grid) with coordinates (x_1, y_1) for the top left corner and (x_2, y_2) for the bottom right corner as rectangle $A = (x_1, x_2, y_1, y_2)$. Its processing consists of the following steps.

Step 1: If A is a single pixel (i.e. $x_1 = x_2$ and $y_1 = y_2$), then it can't be split any further. We let the representative value of A be $r(A) = f(x_1, y_1)$. Then we encode this region as described below and return.

Step 2: For each row (y_1 through y_2) and column (x_1 through x_2) compute the sum and the sum of squares of the values of its elements. This step is done for efficiency of later computations.¹

Step 3: We compute the representative value of this region, $r(A)$, and a quantity $q(A)$ indicative of heterogeneity of this region (as will be described in the next section). If this quality is above the threshold computed during the preprocessing (i.e. if $q(A) > \tau$), then this region should be split further. Otherwise, we encode this region and return.

¹The algorithm can become even more efficient if these are computed during preprocessing step as running sums, but we have not yet implemented this step.

Step 4: Among all partitions of A into two rectangles (along X- or Y-axis), find a partition into B and C that minimizes the value of (2). This can be done in linear time since all the row and column values have been precomputed in Step 1.

Step 5: Repeat Steps 0-4 first on B , then on C .

As can be seen from the above description, our algorithm is a hierarchical divisive partitioning method that chooses the optimal partition at each step.

Representing Regions

Once we find a homogeneous region, we need to represent it efficiently. A simple approach is to represent a rectangle $A = (x_1, x_2, y_1, y_2)$ by five numbers: $x_1, x_2, y_1, y_2, r(A)$. A clear benefit of such representation is that the reconstruction of A is straightforward - one just needs to set the value of all pixels with coordinates (x, y) , such that $x_1 \leq x \leq x_2$ and $y_1 \leq y \leq y_2$, to $r(A)$. The full grid G can then be represented by a sequences of such representations. However, a more efficient representation is possible based on the fact that we have a partition of grid G , for which we know the lengths of the sides, s_x and s_y , and that the regions were processed by our algorithm in a specific order. In this representation we store only the coordinates of top left corner and the $r(A)$ value for each region. The whole grid is stored as a pair s_x and s_y followed by triplet descriptions of the regions, in the order they were produced.

We now describe how the grid can be decoded from such a representation.

Step 1: Construct a grid of size s_x by s_y . Set the value of each pixel to some default, denoting absence of the actual value (for example NaN).

Step 2: Read the sequence of triples describing the regions and reverse it.

Step 3: For each region A in the reversed sequence, described by coordinates (x_1, y_1) , and value $r(A)$, do the following: for each row y_1 to s_y (column x_1 to s_x) move left to right (or top to bottom) as long as the current pixel's value is NaN and change it to $r(A)$. Since the regions are handled in reverse order compared to our Algorithm, the process stops exactly at the boundary of the region.

4. COMPUTATIONAL REQUIREMENTS

We analyzed potential implementation of the Mercury embedded multicomputer based on a Motorola PPC general-purpose processor with 500 MHz processor and AltiVec vector unit. See [4] for detailed parameters of the hardware. Vector unit

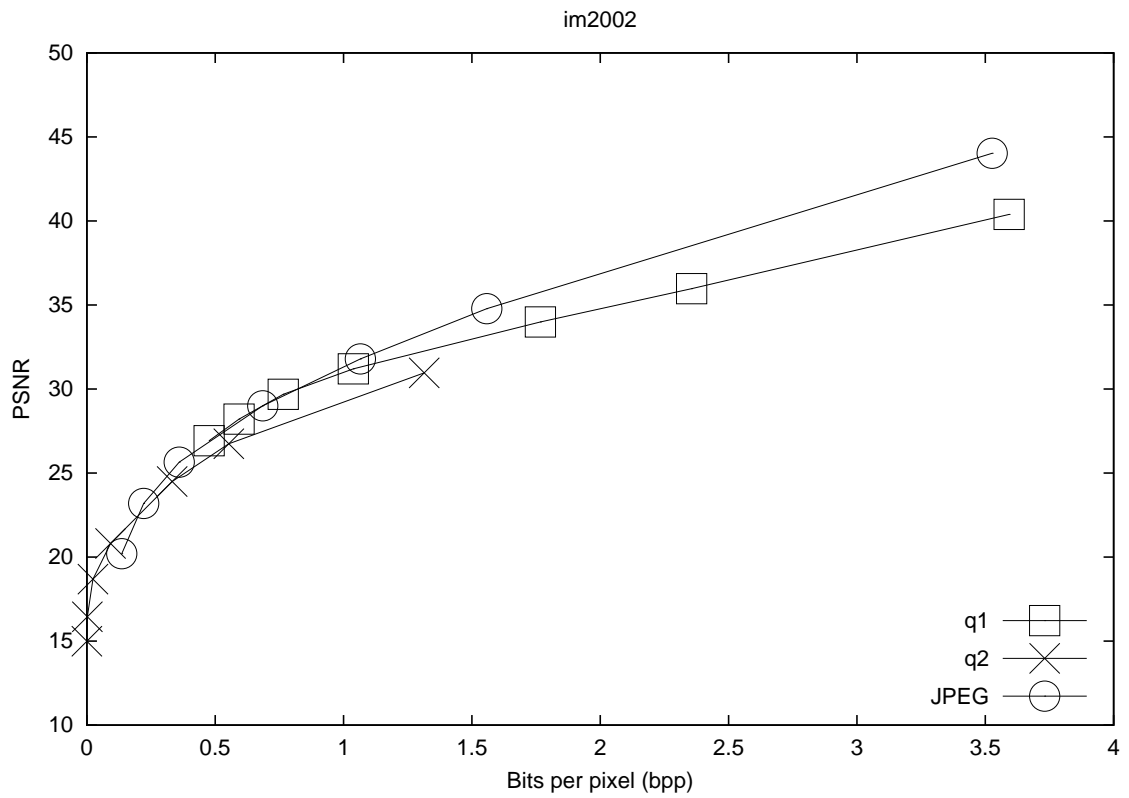


Figure 1. Plot of compression vs. quality, im2002

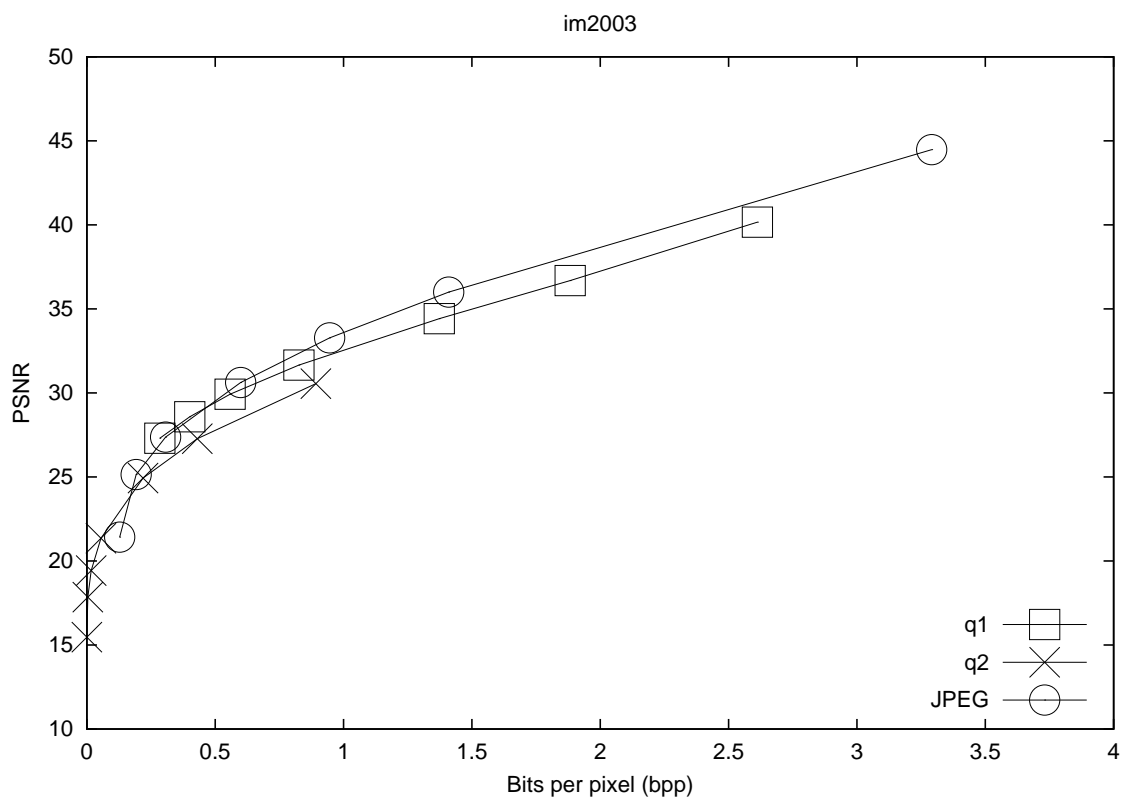


Figure 2. Plot of compression vs. quality, im2003

TABLE 1. Results of our method

PSNR	bpp	NL	$q(A)$	ϵ	Image
40.389	3.593	18.866	1	0.10	im2002
35.971	2.356	18.107	1	0.15	im2002
33.986	1.767	17.528	1	0.20	im2002
31.197	1.038	16.289	1	0.30	im2002
29.685	0.765	15.518	1	0.40	im2002
28.208	0.593	14.857	1	0.50	im2002
26.924	0.476	14.257	1	0.60	im2002
40.160	2.612	18.467	1	0.10	im2003
36.682	1.883	17.819	1	0.15	im2003
34.420	1.373	17.117	1	0.20	im2003
31.654	0.826	15.932	1	0.30	im2003
29.921	0.558	15.024	1	0.40	im2003
28.579	0.401	14.278	1	0.50	im2003
27.305	0.284	13.472	1	0.60	im2003
30.965	1.314	15.749	2	0.10	im2002
26.741	0.553	13.509	2	0.15	im2002
24.499	0.332	12.262	2	0.20	im2002
20.807	0.092	9.692	2	0.30	im2002
18.682	0.024	7.462	2	0.40	im2002
16.450	0.002	3.902	2	0.50	im2002
14.992	0.000	1.000	2	0.60	im2002
30.553	0.892	14.240	2	0.10	im2003
27.272	0.430	12.084	2	0.15	im2003
24.929	0.219	10.548	2	0.20	im2003
21.355	0.055	7.841	2	0.30	im2003
19.424	0.017	5.994	2	0.40	im2003
17.844	0.004	4.556	2	0.50	im2003
15.467	0.000	2.411	2	0.60	im2003

TABLE 2. JPEG results

PSNR	bpp	Quality	Image
44.025	3.527	95	im2002
34.780	1.558	75	im2002
31.795	1.065	50	im2002
28.999	0.685	25	im2002
25.649	0.360	10	im2002
23.186	0.221	5	im2002
20.202	0.136	2	im2002
44.476	3.291	95	im2003
35.996	1.409	75	im2003
33.278	0.946	50	im2003
30.620	0.599	25	im2003
27.369	0.307	10	im2003
25.144	0.192	5	im2003
21.418	0.128	2	im2003

is capable to process 128 bits of streaming data (divided in either 4 floating point numbers, or 8 16-bit integer, or 16 8-bit integers) in one processor clock. Faster processors are currently available for benign environments, but 500 MHz processors are typical for military embedded systems where power and heat are important constraints.

Most of inner-loop computations in the described algorithm are matrix computations that are well suited for vector unit implementation. While a significant speedup can be achieved by using AltiVec C extension, reaching the ultimate performance requires assembly coding that maximizes processor throughput. Mercury provides extensive library of assembly-coded matrix and vector image processing operations, PiXL, assembly-optimized for AltiVec processing ([10]). Most of the inner-loop operations are covered by 8- and 16-bit functions in this library. According to PiXL timings, our algorithm requires $5 + 1.2 * NL$ processor clocks, where NL is average number of partitioning levels on the image. For example, a 10-level algorithm will require 17 clocks. Then, each 500 MHz processor achieves throughput of 30 MBs. Then, a four-processor sub-system of any of scalable Mercury VME systems can process 120 MB/s of a HDTV stream.

5. DATA

We tested the proposed algorithm on two Radio Plasma Imager (RPI) Spectrogram images from NASA Image Project [11]. The RPI instrument is a radar which operates in the radio frequency bands that contain the plasma resonance frequencies characteristic of the Earth's magnetosphere (3 kHz to 3 MHz). By stepping through various frequencies for the transmitted signal, features of various plasma densities can be observed. The RPI instrument contains four 250 m wire antennae. The images display the amplitude of XY antennae as a function time (in milliseconds since 01-Jan-0000 00:00:00.000) and frequency [12].

We selected images dated May 2, 2002 and January 1, 2003. These color images were created for visualization purposes and mask the true values. We analyzed the colormap of the images and extracted the pixel values on the underlying scale. We thus obtained two images, im2002 and im2003, with 256 grey levels that we used for the experiments.

6. EXPERIMENTAL RESULTS

We present results for average representative value (4) and each of the criteria $q_1(A)$ (6) and $q_2(A)$ (7). For each of these settings, we varied parameter ϵ as 0.1, 0.15, 0.2, 0.3, 0.4, 0.5, and 0.6. We note that higher values of ϵ lead to lower quality and higher compression rate.

In order to compare our results with JPEG, we used Independent JPEG Group implementation [1] that takes a quality parameter with values from 0 (lowest quality) to 100 (highest). We used settings 95, 75, 50, 25, 10, 5, 2. We did not use “-optimize” option as this produces slightly smaller files at the cost of much more memory and longer computations.

We used standard measures of image compression to express compression results from our method and JPEG:

$$RMSE(A) = \sqrt{\frac{\sum_{(x,y) \in A} (p(x,y) - r(A))^2}{|A|}} \quad (10)$$

where $p(x, y)$ is the value of pixel (x, y) in the original image, and

$$PSNR(A) = 20 \times \log_{10}\left(\frac{255}{RMSE(A)}\right), \quad (11)$$

where PSNR is a measure of quality of compression. PSNR is higher for better approximations of the original image.

“Bits per pixel” (bpp) measure is used to describe degree of compression. For JPEG, we computed compression ratio by dividing the size of the compressed file (in bits) produced by JPEG by the number of pixels. For our method, we computed compression ratio under assumption that each rectangle can be encoded by 28 bits: 10 bits for each coordinate of the top left corner of a rectangle A and 8 bits for the value $r(A)$. This assumption holds true for images 1024×1024 or smaller. Then $bpp = \frac{28 \cdot K}{|G|}$, where K is number of rectangles and $|G|$ is the size of the image.

We also computed the average number of levels NL processed by our method for each pixel since NL can be used to estimate computational complexity of our method (as described in Section on Computational Requirements below).

The measures described above are presented for JPEG algorithm in Table 2 and for our method in Table 1. We also plotted the graphs of “bpp vs. PSNR”. These are displayed in Figures 1 and 2.

7. DISCUSSION AND FUTURE RESEARCH

Our results look promising. The proposed algorithm achieves compression ratios comparable with JPEG at potentially lower computational cost and complexity of the algorithm. The algorithm performs especially well for low-bit compression ratios that are of interest in many DoD and NASA applications. Our algorithm can be further improved by applying standard pre- and post-processing steps, such as lossless compression of the results. In addition, our algorithm works solely in the pixel space and thus does not introduce artifacts typical for spectral methods that make consequent processing of the images too difficult.

For future research, we plan to experiment with alternative selection of functions $r(A)$ and $q(A)$ that greatly influence achievable compression ratio and quality. Furthermore, instead of representing a rectangle by a piece-wise constant function, as we have done, it may be advantageous to consider linear or Gaussian representations. While they would require additional storage for a rectangle, they might improve the quality of the region and prevent it from splitting, thus saving space elsewhere.

REFERENCES

- [1] Independent JPEG Group. [Online]. Available: <http://www.ijg.org/>
- [2] G. Jaffe, “Military feels bandwidth squeeze as the satellite industry sputters,” *Wall Street Journal*, April 2002.
- [3] O. of the Secretary of Defense, “Unmanned aerial vehicles roadmap,” December 2002. [Online]. Available: http://www.acq.osd.mil/usd/uav_roadmap.pdf
- [4] S. Streltsov, “Multiple hypothesis tracking,” in *Proceedings of the KIMAS*, 2003.
- [5] “Mercury computer systems defense electronics overview.” [Online]. Available: http://www.mc.com/defense_electronics/
- [6] S. Lloyd, “Least squares quantization in PCM,” *IEEE Transactions on Information Theory*, vol. 28, pp. 128–137, 1982.
- [7] W. D. Fisher, “On grouping for maximum homogeneity,” *J.Am.Stat.Assoc.*, vol. 53, pp. 789–798, 1958.
- [8] J. MacQueen, “Some methods for classification and analysis of multivariate observations,” in *Proceedings of the Fifth Berkley Symposium on Mathematical Sciences and Probability*, 1967, pp. 281–297.
- [9] B. Mirkin, “Approximation clustering as a framework for solving challenging problems in processing massive datasets,” in *DIMACS MiniWorkshop Exploring Large Data Sets Using Classification, Consensus, and Pattern Recognition Techniques*, 1997.
- [10] “Mercury computer systems pixl library.” [Online]. Available: <http://www.mc.com/search/allproducts.cfm>
- [11] NASA, 2003. [Online]. Available: <http://image.gsfc.nasa.gov/rpi/>
- [12] J. Green, “Personal communication,” June 2003.