

An Efficient MAP Classifier for Sensornets

Zille Huma Kamal, Ajay Gupta, Leszek Lilien and Ashfaq Khokhar[†]

Dept. of Computer Science, Western Michigan University, Kalamazoo, MI

[†]Dept. Computer Science, Dept. Electrical & Computer Eng., University of Illinois, Chicago
{zkamal, gupta, lilien}@cs.wmich.edu

[†]ashfaq@uic.edu

Abstract. The classification phase is computationally intensive and frequently recurs in tracking applications in sensor networks. Most related work uses traditional signal processing classifiers, such as Maximum A Posterior (MAP) classifier. Naïve formulations of MAP are not feasible for resource constraint sensornet nodes. In this paper, we study computationally efficient methods for classification. We propose to use one-sided Jacobi iterations for eigen value decomposition of the covariance matrices, the inverse of which are needed in MAP classifier. We show that this technique greatly simplifies the execution of MAP classifier and makes it a feasible and efficient choice for sensornet applications.

Keywords. Classification, Jacobi Iterations, Sensor Networks, Collaborative Processing

1. Introduction

Sensornets have been envisioned as a cost effective paradigm for monitoring, controlling or sensing in military, agricultural, or commercial applications. In any of these applications, the fundamental criterion for detecting the presence or absence of an object or an environmental condition requires ‘sensing’ of the features or modalities present in the environment. After detecting a stimulus, classification algorithms process the sensed data to categorically determine and identify the event (stimulus), results of this classification phase can be used to decide whether a reaction/response is warranted. In monitoring/surveillance applications—[1], [2], [3], [4], to mention a few, the classification phase is followed by a tracking phase, where the sensornet nodes work collaboratively to track the stimulus, which could be an enemy combat vehicle or the spread of hazardous chemicals in the environment.

The classification process is a computationally intensive process, repeatedly initiated for every measurable disturbance in the environment that the sensornet is trained to observe. A traditional approach to classification in sensornets is to use classifiers that are commonly employed in signal-processing applications, such as Linear Vector Quantization, Support Vector Machines (SVM), k-Nearest Neighbor (kNN) and Maximum Likelihood (ML), among which ML is considered most feasible for sensornets due to its minimal storage and computation requirements [4].

When all categories are equally probable, the Maximum A Posterior (MAP) classifier is the same as ML ([22], [23]). In most cases, we can easily hold and justify this

assumption. Thus, we critically review the computations and communications required in a distributed MAP classifier and its feasibility in resource constraint sensor-net applications.

Computation of the MAP classifier requires the inverse of the covariance matrices that represent the pre-determined categories. Often inverse computation is avoided, due to the numerical instability [18]. Instead, techniques that approximate the inverse, decompose the inverse computation, or perform orthogonal transformations, are preferred [16]. Therefore, we instigate the use of stable, parallelizable and efficient one-sided Jacobi iterations and transform the problem of computing the inverse of the covariance, into computing its Eigen Value Decomposition (EVD). We will show that the substitution of EVD of covariance matrix in place of its inverse, in MAP can greatly reduce the cost of this classifier. To the best of our knowledge, using Jacobi in MAP has not been considered previously.

The main contributions of this paper are to show how to use one-sided Jacobi iterations in sensor-nets and consequently minimize the cost of executing the MAP classifier in sensor networks.

We begin by summarizing related work in this area in section 2. In section 3 we give an overview of the MAP classifier. In Section 4, we review the one-sided Jacobi iterations and its computation and communication costs. We present computational and communicational costs of MAP with Jacobi and MAP with LU decomposition (previous work [21]) in Section 5 and compare the power consumption in Section 6. We conclude with our findings and contributions in Section 7.

2. Related Work

Authors in [1], [3] and [4] have proposed the use of classifiers such as kNN and MAP in classification applications of sensor-nets. However, their test-bed and sensing platform is composed of WINS 3.0 Sensoria [9], [10] nodes that are more powerful in terms of energy, processor speed and power, and memory capacity, compared to the more constraint Crossbow's MICA2 Motes [11], Intel Motes [14] or the envisioned dust size COTS Dust [12], [13]. In this paper, we use *sensors* or *nodes* interchangeably to generalize small resource constrained motes, like MICA2 or Intel Motes. An analytical study conducted in [5] concluded that there was an acute trade-off between accuracy and efficiency of classifiers in sensor-nets and it became apparent that traditional classifiers were computationally and communicationally intensive and unpractical for today's resource constraint sensor-net (mote-like) nodes.

Some alternate approaches to conventional classification include novel, computationally efficient influence field patterns [7] to classify objects, but its accuracy is shown to be directly dependent on the reliability of the underlying sensor-net, a tough characteristic to build into a *dynamic* sensor-net. Gu et. al. [24] considers hierarchical classification, first on node level, then group level followed by a base level, which has been shown to be feasible and accurate for sensor-nets. In [6], power conservation is built into sensor-net surveillance application by differentiating the sensing coverage of the region monitored based on the proximity to the highly sensitive area, e.g. the army base station in a military application.

Other alternate techniques [8] use mobile agents (robots or humans) to gather and transfer data from sensornet to a more powerful command/base station for processing. While this approach may be appropriate in certain situations, it seems infeasible for applications that are envisioned to be deployed in monitoring impermeable, insecure regions.

Though, it seems that traditional signal processing classifiers, like MAP, are inappropriate for sensornets that are resource constraint [24], in terms of processing power, small storage and absence of floating point hardware [24], [25], [29]. Our goal is to avoid reinventing the wheel and adapt these existing tested classifiers, specifically, the MAP classifier, and try to make it feasible for these sensornets.

3. Background

3.1 Classification

Classification consists of three phases: training, testing and deployment. In the *training phase*, the sensors in the sensornet are exposed to N known sample event feature matrices from each of the k predetermined categories. The user interactively classifies the events for the sensors. An *event* is an $f \times d$ feature matrix, where f is the number of features monitored in the environment and d is the temporal dimension. Based on these sample feature matrices, each sensor can locally compute the *mean* (μ) and *covariance* (δ) matrix for a category as in Eq. 1 and 2. $M_{i,j}^\alpha$ is the event feature matrix from the training phase (indicated by α) at sensor i for category j , for $1 \leq i \leq n_0$, where n_0 is the total number of sensors in the sensornet and $1 \leq j \leq k$.

$$\mu_{i,j} = \frac{1}{N} \sum_{\alpha=1}^N M_{i,j}^\alpha. \quad (1)$$

$$\delta_{i,j} = \frac{1}{N-1} \sum_{\alpha=1}^N (M_{i,j}^\alpha - \mu_{i,j})(M_{i,j}^\alpha - \mu_{i,j})^T. \quad (2)$$

In the *testing phase*, probability of false positives and false negatives are computed to determine a belief probability for the sensor's accuracy, by allowing the *sensors to classify the test sample events*. In the deployment phase, sensors are deployed in the region to be monitored, unmonitored themselves, and repeat the classification process to decide whether a detected object is of interest or not.

3.2 MAP Classifier

In this paper we study the Maximum A Posterior (MAP) classifier, the computation of which may be complex for resource constraint sensornets. However, by removing terms that are constant across the categories, computation of MAP can be simplified ([5], [22], [23]), to the computation of the *Mahalanobis distance* [22] between the mean of a category and the event/object detected (Eq. 3). The detected object is classified into the category that minimizes the Forbenius norm [26] of the Mahalanobis

distance (Eq. 4), where $M_{i,j}$ represents event recorded at sensor i for some *unknown* category j , and $\|MAP_{i,j}\|_F$ represents the Frobenius norm of the $d \times d$ MAP matrix. Here, β is used to indicate event feature matrices that are collected during the testing phase, however, same computations are performed during deployment phase.

$$MAP_{i,j}^\beta = (M_{i,j}^\beta - \mu_{i,j})^T \delta_{i,j}^{-1} (M_{i,j}^\beta - \mu_{i,j}). \quad (3)$$

$$\|MAP_{i,j}^\beta\|_F = \sum_{p=1}^d \sum_{q=1}^d MAP_{i,j}^\beta(p,q). \quad (4)$$

Generally, sensornet ‘surveillance’ applications decompose the region monitored into logically or physically disparate cells, clusters or sub regions. We design clusters of size d sensors where d is the larger dimension of the events. These dense clusters enable us to exploit parallelism to run the computationally-intensive MAP quickly, efficiently and in a load balanced manner.

We assume for brevity that the distributed FFT ([27], [34]) for sensornets can be extended to perform efficient d -point temporal processing of the $f \times d$ matrix, distributively across the d sensors. So, each sensor holds a column of the mean and covariance matrices, as in Fig. 1, which illustrates the clustering and data distribution.

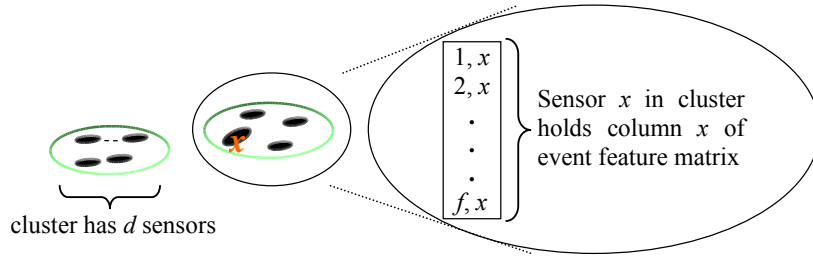


Fig. 1. Data distribution in a cluster node

The most complex computation in Eq. 3, is finding the inverse of $\delta_{i,j}$ for all k categories. One might consider offloading the computation of $\delta_{i,j}^{-1}$ to a more powerful node such as a personal computer. However, this would inhibit the sensornet from dynamically updating $\delta_{i,j}$ and its inverse that could have increased the accuracy of the classification. To make MAP feasible for sensornets, we must find fast, stable and efficient algorithms for the computation or decompose it into smaller less computationally intensive parts.

Traditionally, computation of inverse of an $f \times f$ matrix requires $O(f^3)$ operations [28], and is slow, inefficient, unbalanced, serial and tedious. Instead, we consider the eigen value decomposition (EVD) of $\delta_{i,j}$, such that it is decomposed into its eigenvectors and eigen values (Eq. 5), where Λ is a diagonal matrix, with eigen values on

the diagonal and E is the matrix, whose columns are eigen vectors. By substituting Eq. 5 in Eq. 3, we get Eq. 6.

$$\delta_{i,j} = E\Lambda E^T \Rightarrow \delta_{i,j}^{-1} = E\Lambda^{-1}E^T. \quad (5)$$

$$\begin{aligned} MAP_{i,j} &= (M_{i,j} - \mu_{i,j})^T E\Lambda^{-1}E^T (M_{i,j} - \mu_{i,j}) \\ \Rightarrow MAP'_{i,j} &= (M'_{i,j} - \mu'_{i,j})^T \Lambda^{-1} (M'_{i,j} - \mu'_{i,j}) \end{aligned} \quad (6)$$

$$MAP_{i,j} = \frac{(m'_{11} - \mu'_{11})^2}{\lambda_1} + \frac{(m'_{21} - \mu'_{21})^2}{\lambda_2} + \dots + \frac{(m'_{f1} - \mu'_{f1})^2}{\lambda_f} + \dots + \frac{(m'_{fd} - \mu'_{fd})^2}{\lambda_f}. \quad (7)$$

Matrix algebra says, $(M'_{i,j} - \mu'_{i,j}) = ((M_{i,j} - \mu_{i,j})^T E)^T = E^T (M_{i,j} - \mu_{i,j})$ and multiplication of a matrix with Λ^{-1} is just a scalar multiplication of column x of the matrix with $1/\lambda_x$, which is the reciprocal of the diagonal entry in column x of Λ . Using these rules, Eq. 6 becomes the equation of a hyper-ellipse (Eq. 7), centered at μ' [22], and $m'_{p,q}, \mu'_{p,q}$ are the p,q entry in matrix $M'_{i,j}$ and $\mu'_{i,j}$ and λ_p is the p^{th} eigen value in Λ . Eq. 7 is simpler than Eq. 6, as matrix multiplication is eliminated.

We use the inherently parallel one-sided Jacobi iterations for EVD of the covariance matrix. It has been shown that Jacobi parallelizes well [20], hence we do not include any speedup or parallelization efficiency analysis. We show how one-sided Jacobi can be used efficiently for MAP and its feasibility in mote-like sensor networks.

4.0 One-sided Jacobi Iterations

Jacobi Iterations gained renewed popularity with advancements in parallel computing [19] and are used for eigen (or spectral or singular) valued decomposition (EVD/SVD) of a symmetric covariance matrix. The goal is to apply similarity transformations to zero-off (eliminate) the off-diagonal entries. The multipliers used for the transformations are accumulated, and yield the eigen vectors of the covariance matrix and the remaining diagonal entries represent the eigen values. The inherent parallelism of one-sided Jacobi iterations makes it a better choice than its two-sided Jacobi counterpart [15], [20], for our distributed sensor networks. This parallelism will enable a faster and more robust application.

The similarity transformation to eliminate entry p,q of the matrix, termed a *step* [20], only affects columns p and q of the matrix, and can be performed in parallel with the step for entry $a,b, \forall a \neq p$ and $b \neq q$. This implies $f/2$ steps can be performed in parallel, to eliminate the off-diagonal entries of an $f \times f$ matrix. A *sweep* [20] contains $f-1$, if f is even, or f , if f is odd, steps that are required to eliminate all off-diagonal entries. After every step, an exchange of columns (or rows) must occur, termed *transition* [20].

Generally, more than one sweep may be required, since a step may introduce a non-zero entry in a previously zeroed off entry. Typically, $\log_2 f$ sweeps converge a

symmetric matrix, faster convergence can be achieved with a threshold ϵ , ([15], [20]), that determines whether an entry is considered for elimination.

The order in which the transformations are applied also affects the rate of convergence [20]. Various orderings, such as row-cyclic [15] and block recursive (BR) [20] have been shown to achieve high convergence. Without loss of generality, we will use BR ordering, whose efficient implementation on hypercube has been shown in [20]. Therefore, for simplicity, we abstract a hypercube topology from the sensornet for each cluster, such that for the $f \times f$ covariance matrix, we have a hypercube with $\log_2 f$ dimensions and $f/2$ sensors.

4.1 Block Recursive (BR) Ordering

Initially in BR ordering, we distribute two columns of $\delta_{i,j}$ to each sensor, the smaller index value column is called *top* and the larger becomes the *bottom* column, succeeded by three phases delineated as: (i) Rotation phase, a series of parallel steps and transitions, (ii) Top and Bottom Exchange (TBX) phase, a step followed by a transition, and (iii) Last Exchange (X) phase, the last step and transition of a sweep. The Jacobi procedure is presented in the program code, `One_Sided_Jacobi` and the computations required for a step are delineated in the program code, `Step`.

Initially, we let, $S_0' = \delta_{i,j}$, $S_0 = \delta_{i,j}$ and $U_0 = I$ (the identity matrix). If we assume that the iterations converge in $\log_2 f$ sweeps, the eigen vector matrix E will be stored in the U_i 's, i.e. $E = U_{\log_2 f}$ and similarly $\Lambda = S_{\log_2 f}$ [20]. The step computations required in the Rotation, TBX and X phases, iteratively update the initial matrices, S_0' and $U_0 = I$ and consequentially S_k is updated as in line 10, in `One_Sided_Jacobi` procedure.

<pre> One_Sided_Jacobi ($\delta_{i,j}$, ϵ) (1) $S_0' = \delta_{i,j}$; $U_0 = I$; (2) for $k = 1$ to $\log_2 f$ (3) //perform sweep (4) while (p, q in $S_k > \epsilon \bullet \bullet$) (5) //compute $S_k' & U_k \rightarrow$ Step (6) Rotation; TBX; X (9) end while (10) $S_k = U_k^T S_k'$ (11) end for end procedure </pre>	<pre> Step ($S_k, S_k', U_k, p, q, \epsilon$) (1) $\theta = \frac{S_k(q,q) - S_k(p,p)}{S_k(p,q)}$ (2) $c_{p,q} = \cos \theta$; $s_{p,q} = \sin \theta$ (3) $S_{k+1}'(\bullet, p) = c_{p,q} * S_k'(\bullet, p) + s_{p,q} * S_k'(\bullet, q)$ (4) $S_{k+1}'(\bullet, q) = -s_{p,q} * S_k'(\bullet, p) + c_{p,q} * S_k'(\bullet, q)$ (5) $U_{k+1}(\bullet, p) = c_{p,q} * U_k(\bullet, p) + s_{p,q} * U_k(\bullet, q)$ (6) $U_{k+1}(\bullet, q) = -s_{p,q} * U_k(\bullet, p) + c_{p,q} * U_k(\bullet, q)$ end procedure </pre>
--	---

The dimensions used to exchange the columns in a transition are generated as $D_1^{BR} = \langle 0 \rangle$ and $D_r^{BR} = \langle D_{r-1}^{BR}, r-1, D_{r-1}^{BR} \rangle$ [20], where, D_r^{BR} is the BR sequence of dimension(s) used for transitions in rotation phase r , D denotes the dimensions used for column exchange and is different from d used earlier to denote the points of temporal processing of each feature. The dimension used for transitions of successive sweep, j , is a permutation of the dimensions used in the respective rotation of the first

sweep, simply generated as, $(k - j) \bmod (\log_2 f)$, for every dimension k , in the sequence D_r^{BR} , for rotation r in the first sweep [20].

4.2 Adapting one-sided Jacobi to Sensornets

The matrix multiplication required to update S_k (line 10, procedure One_Sided_Jacobi) may not be feasible for our sensornet, since the columns of S_k are distributed and communication is expensive. Furthermore, if all we need is $E = U_{\log_2 f}$ and $\Lambda = S_{\log_2 f}$, then we should avoid updating S_i (lines 3–4, procedure Step). Then, the computation of θ (line 1, procedure Step), can be formulated as the inner product of the columns of U and S (Eq. 8) ([30],[31]), where u_p^k and s_p^k are

the p^{th} columns of U_k and S_k , respectively, and $\langle u^p, s^p \rangle = \sum_{i=1}^f u_i^p \cdot s_i^p$.

$$\tan 2\theta = \frac{2\langle u_p^k, s_q^k \rangle}{(\langle u_p^k, s_p^k \rangle - \langle u_q^k, s_q^k \rangle)} = \Psi. \quad (8)$$

$$\theta = \frac{\tan^{-1}(\Psi)}{2}. \quad (9)$$

Then, by letting $t_{p,q} = \tan \theta$, we have $c_{p,q} = \cos \theta = \frac{1}{\sqrt{1+t_{p,q}^2}}$

and $s_{p,q} = \sin \theta = c_{p,q} * t_{p,q}$, which will replace lines 1–2 in procedure Step and eliminate line 10 in procedure One_Sided_Jacobi. This way, the entire step can be computed locally, as the sensors hold the necessary columns. Moreover, to overcome the absence of hardware support for floating point operations, and to conserve processor cycles we can maintain a small finite table that stores values of θ and its trigonometric cosine and sine values. This approach has been shown to achieve accurate results in [17] and we refer readers to it for a detailed study of this method.

4.3 Computational and Communicational Cost of One-sided Jacobi Iterations

The preceding sections review classification and one-sided Jacobi iterations, indicating the adaptations made to them to make them feasible for sensornets. We now derive computational and communicational costs of one-Sided Jacobi and MAP based on these discussions.

4.3.1 Cost of a Sweep - Steps and Transitions in Rotation, TBX and X Phases

The computation costs incurred in a step are: computation of Ψ (Eq. 8), table lookup operation, scalar-vector multiplications and vector-vector additions (lines 5–6,

procedure `STEP`), that incur a total cost $12f + 6f + O(\log_2 m) \approx 18f$ local operations without *any* additional communication, where m is the number of θ values stored in the table for lookup.

There are $\log_2 f$ rotations in a sweep and the number of steps and transitions in a Rotation phase, r , are $2^r - 1$ [20]. Then, the total number of steps and transitions in a sweep from the Rotations are $\sum_{r=1}^{\log_2 f} 2^r - 1 = \frac{2^{\log_2 f + 1} - 1}{2 - 1} - \log_2 f = 2f - 1 - \log_2 f$.

There is a TBX phase after every Rotation phase, with one step and transition in it, so there are $\log_2 f$ steps and transitions from the TBX phases of a sweep. Since, there is only one X phase in a sweep, the *total number of steps and transitions* in a sweep are $2f - 1 - \log_2 f + \log_2 f + 1 = 2f$.

4.3.2 Total Cost for One-sided Jacobi

The step is the only and purely local computation required in the one-sided Jacobi iterations. The cost of a sweep is $(2f)(18f) = 36f^2$ at a sensor. So, the total *computational* cost incurred after $\log_2 f$ sweeps is $(36f^2)(\log_2 f) = 36f^2 \log_2 f$ across $f/2$ sensors of the d sensors in a cluster of the sensornet. The local computation minimizes communication costs thereby optimizing overall MAP computation costs. Each of the $f/2$ sensors involved in the Jacobi computation do an equal amount of work and hence expend equal energy which increases network lifetime.

In a transition, each sensor incurs communicational costs to send and receive a message, which is a column of the matrix. A TinyOS message payload is 29 bytes long [32], therefore, a message with f elements, assuming an element is stored in 4 bytes, requires $\left\lceil \frac{4f}{29} \right\rceil$ messages to be sent and received. Thus, the total *communicational* cost incurred across the sensors is the total number of messages sent and received in $\log_2 f$ sweeps i.e. $2f \left\lceil \frac{4f}{29} \right\rceil \log_2 f = 2f \log_2 f \left(\frac{4f}{29} + 1 \right)$.

5.0 Cost of Maximum A Posterior (MAP) Classifier

5.1 MAP with Jacobi (MAP-J)

Once $\delta_{i,j}$ converges, MAP can be computed as presented in Eq. 7, for which we first redistribute the columns held by the $f/2$ sensor nodes that were participating in the one-sided Jacobi iterations. This way, sensor node i holding column i of $M_{i,j}$, now also holds column i of S_k^{-1} ($=\Lambda^{-1}$) and U_k , where $k=\log_2 f$. Then, each of the f sensors can concurrently compute one of the fraction entities of Eq. 7 locally. This requires 3 arithmetic operations for each of the f entries in a column of the matrix, followed by summation of all local fractions, which implies a total of $4f-1$ (i.e. $3f+f-1$) operations at each of the d sensors in a cluster.

Thus, the MAP classifier incurs a total *computational* cost of $(4f-1)d = (4fd-d)$. To efficiently aggregate these distributed fractions, we can use a tree-like structure so that all entries can be aggregated in $\log_2 d$ levels, where each level requires transmission and reception of one message. Thus, the total messages sent and received are d , the total *communicational* cost of MAP with Jacobi (MAP-J).

5.2 MAP using LU (MAP-LU)

Recall the inverse of the covariance matrix in MAP can also be computed using the popular LU decomposition technique. Our earlier work in [21] presents this approach in detail. In this section, we just restate the results briefly to facilitate comparison with MAP classifier using one-sided Jacobi. The LU decomposition is used to decompose the covariance matrix into its Lower (L) and Upper (U) triangular matrices. Eq. 3 can now be formulated as Eq. 10, where, D is the diagonal entries of U .

$$\begin{aligned} MAP_{i,j} &= (M_{i,j} - \mu_{i,j})^T LD^{-1}L^T(M_{i,j} - \mu_{i,j}) \\ \Rightarrow MAP'_{i,j} &= (M'_{i,j} - \mu'_{i,j})^T D^{-1}(M'_{i,j} - \mu'_{i,j}) \end{aligned} \quad (10)$$

However, this cannot be further simplified like we did in Eq. 7. This is because, LU decomposition is not a similarity transformation, like one-sided Jacobi that can preserve the eigen values [20] of the original matrix, and allow the simplification of Eq. 7.

Therefore, the cost of MAP with LU (MAP-LU) can be stated as follows:

let $Z = (M_{i,j} - \mu_{i,j})^T L$ and $(M_{i,j} - \mu_{i,j})L^T = Z^T$, so computing Z incurs $2f^2d$ operations (i.e. fd subtractions and fd dot products) and $\left\lceil \frac{4f}{29} \right\rceil = O(f)$ message exchanges

for interchanging columns of L. The cost of computing the norm of MAP (Eq. 4) is $2d^2f-d^2$ for the d^2 dot products and the communicational cost for the norm

is $\left\lceil \frac{4d}{29} \right\rceil = O(d)$.

Thus, over the d sensor nodes of a cluster, the total computational cost of computing MAP-LU is $O(2d^2f + 2df^2 - d^2) = O(d^2f)$ with a communicational cost of $O(d+f) = O(d)$, since $d \gg f$.

6.0 Power Consumption Comparison

We now compare the power consumption of MAP-J with MAP-LU. Various authors, ([29], [33]), to mention a few have studied the average power consumption of a Mica mote-type sensor node in various modes. We use the power ratings presented in [29], where the power consumption for transmitting, receiving and computing (only CPU operational) is approximately 10 mA, 7 mA and 8mA, respectively. From Section 5.1, MAP-J expends a total of $(4fd-d) \times 8 + d \times 10 + d \times 7 = 32fd + 9d$ mA and simi-

larly, MAP-LU, consumes $16d^2f - 8d^2 + 16f^2d + \frac{68(f+d)}{29} + 34$ mA. These power consumptions are plotted in Fig. 2.

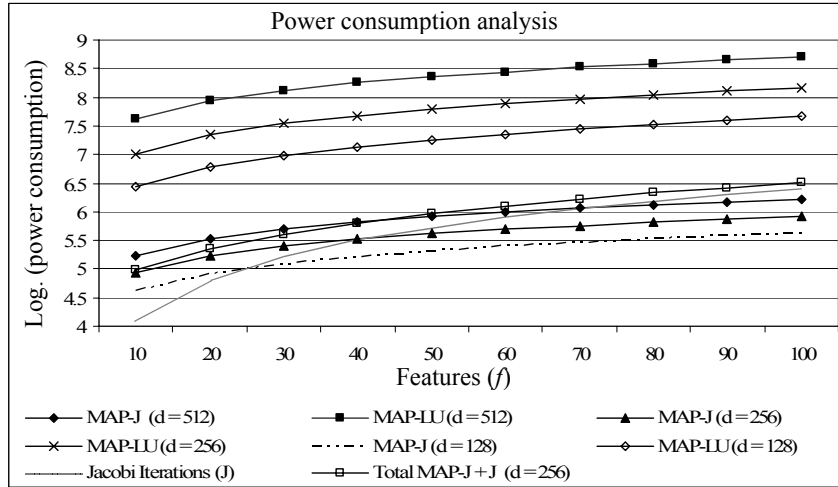


Fig. 2. Comparing power consumed by MAP-J, MAP-LU, with varying dimensions of temporal processing (d), cost of one-sided Jacobi with BR (J) and total cost of MAP-J + J.

It is evident that there are orders of magnitude saving in power consumption of MAP-J with MAP-LU, irrespective of temporal processing points (d). This is because EVD allows the simplifications of Eq. 7 that enables faster computation. Principal Component Analysis (PCA) [26] type of techniques can also be used in MAP-J so that only the first few principal or *significant* eigen values are computed and used and not all f . Although this is not further explored in this paper, it can be seen that PCA will only speedup the MAP-J computation more but at the possible risk of reduced accuracy.

Fig. 2 also shows that cost of computing Jacobi (J) is not drastically expensive, as the total cost of MAP-J and cost of Jacobi is still magnitudes lower than MAP-LU. Furthermore, typically, 20-40 features are monitored, in which range, cost of Jacobi (J) does not exceed MAP-J, for $d \geq 256$. Most importantly, note total power consumption of MAP and Jacobi Iterations (MAP-J + J) is still magnitudes lower than cost of MAP-LU alone, irrespective of the temporal processing dimension, and not taking into account cost of LU, which can only increase costs.

It is also interesting to note that we were able to overcome floating point hardware limitation without any explicit software modules with MAP-J. This may not be the case for MAP-LU, as at the least floating point emulation software would be needed.

The significant difference of MAP-LU and MAP-J is the reduction in total cost by a factor of $O(d)$, which is typically in the order of 512. Also, the computations have been simplified to vector additions, rather than the more complex vector multiplications of MAP-LU.

7.0 Conclusion

We presented a feasible, practical and efficient method for executing the traditional Maximum A Posterior (MAP) classifier distributively, in resource constraint sensor-nets. We do so by simplifying the computation of the MAP classifier into an equation of a hyper-ellipse (Eq. 7) ([23],[20]). This greatly reduces the computations required and significantly cuts down the communication cost, which occur when the MAP classifier uses the traditional approach of LU decomposition to overcome computation of inverse of covariance matrix. Not only are the one-sided Jacobi iterations efficient but they are also more stable than LU decomposition [28]. Furthermore, we do not have to compromise with any hardware limitations, with the one-sided Jacobi iterations.

We have thus shown feasibility of executing traditional signal processing classifiers such as MAP on resource constraint sensor-nets. Our future work includes, analyzing the accuracies of these methods for classification and effect of PCA on the accuracy of one-sided Jacobi iterations.

Acknowledgements. This research was supported in part by the NSF, under grants ACI-0000442, ACI-0203776, IIS-0242840 and MRI-0215356, by the Dept. of Education grant R215K020362, and a Congressional Award, administered by the US Dept. of Education, Fund for the Improvement of Education. The authors would also like to acknowledge Western Michigan University for its support and contributions to the WiSe (Wireless Sensor-nets) Laboratory, Computational Science Center and Information Technology and Image Analysis (ITIA) Center. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the funding agencies or institutions.

References

- [1] D. Li, K. Wong, Y. Hu, and A. Sayeed, Detection, Classification, tracking of targets in micro-sensor networks. *IEEE Signal Processing Magazine*, March 2002.
- [2] A. D'Costa and A. M. Sayeed, Collaborative signal processing for distributed classification in sensor networks. *Lecture Notes in Computer Science* (Proceedings of IPSN'03), (Springer-Verlag, Berlin Heidelberg), pp. 193–208, (F. Zhao and L. Guibas (eds.), Apr'03).
- [3] M. Duarte and Y. H. Hu, Distance Based Decision Fusion in a Distributed Sensor Network. *International Symposium on Information Processing in Sensor Networks* (IPSN) 2003.
- [4] Marco Duarte and Yu-Hen Hu, Vehicle Classification in Distributed Sensor Networks. *Journal of Parallel and Distributed Computing*, Vol. 64 No. 7, 2004.
- [5] Z. H. Kamal, M. A. Salahuddin, A. Gupta, M. Terwilliger, V. Bhuse, and B. Beckmann, Analytical Analysis in Decision and Data Fusion. *Proc. of Conference on Embedded Systems and Applications*, June 2004.
- [6] T. Yan, T. He, and J. Stankovic, Differentiated Surveillance for Sensor Networks. *In Proc. of 1st ACM Conference on Embedded Networked Sensor Systems*, L.A., CA, Dec 2003.
- [7] A. Arora, P. Dutta, S. Bapat, V. Kulathumani, H. Zhang, V. Naik, V. Mittal, H. Cao, M. Demirbas, M. Gouda, Y-R. Choi, T. Herman, S. S. Kulkarni, U. Arumugam, M. Nesterenko, A. Vora and M. Miyashita, A Line in the sand: A wireless sensor network for target detection, classification, and tracking. *Computer Networks (Elsevier)*, 2004.
- [8] Y. Xu, and H. Qi, Distributed Computing Paradigm for Collaborative Signal and Information Processing in Sensor Networks. *J. of Parallel and Distributed Computing*, Aug 2004.
- [9] Sensoria Corporation, last accessed May 12, 2006, sensoria.com

- [10] WINS 3.0 Sensing Platform, sensoria.com/pdf/WINS-3.0-Wireless-Sensing-Platform.pdf
- [11] Crossbow Technology Incorporated, last access May 12, 2006, xbow.com
- [12] Dust Networks, last accessed May 12, 2006 www-bsac.eecs.berkeley.edu/archive/users/warneke-brett/index.html
- [13] B. Warneke, M. Last, B. Leibowitz, K.S.J. Pister, Smart Dust: Communicating with a Cubic-Millimeter Computer. *IEEE Computer*, Computer Society, Piscataway, NJ, Jan. 2001.
- [14] Intel Corporation, 2006, last accessed May 12, 2006. intel.com/research/exploratory/motes.htm
- [15] D. Gimenez, V. Hernandez, R. Geijn, A. Vidal, A Jacobi Method by Blocks on a Mesh of Processors. *Concurrency: Practice and Experience*, Vol 9 Issue 5, 1997 John Wiley & Sons.
- [16] R. Brent, Parallel Algorithms in Linear Algebra. *Proc. 2nd NEC Research Symp., Aug '91*.
- [17] T.J. Herron, K.M. Reddy, R. Garg, and K. Devanahalli, Eigen Decompositions of Covariance Matrices on a Fixed Point DSP. *National Conference on Communication*, Jan 31 - Feb 2, 2003, Indian Institute of Technology, Madras, India
- [18] D. Bertsekas, and J. Tsitsiklis, Parallel and Distributed Computation – Numerical Methods. Prentice-Hall, Inc., Englewood, New Jersey, 1989.
- [19] B. B. Zhou, R. P. Brent and M. H. Kahn, Efficient one-sided Jacobi algorithms for singular value decomposition and the symmetric eigen problem. *Proc. IEEE First International Conference on Algorithms and Architectures for Parallel Proc.*, IEEE Press, 1995, 256-262.
- [20] D. Royo, A. Gonzalez, and M. Valero-Garcia, Low Communication Overhead Jacobi Algorithms for Eigenvalues Computation on Hypercubes. *J. Supercomputing*, 14(2), Sept'99.
- [21] Z. H. Kamal, A. Gupta, L. Lilien, and A. Khokhar, Classification Using Efficient LU Decomposition in Sensornets. *Proceedings of WSN 2006*, July 2006.
- [22] C. W. Therrien, Decision Estimation and Classification. An Introduction to Pattern Recognition and Related Topics. John Wiley & Sons, NYC, NY, 1989
- [23] Online Resource, Last Accessed May 9, 2006 prof.udec.cl/~gabriel/tutoriales/rsnote/cp11/cp11-7.htm
- [24] L. Gu, D. Jia, P. Vicaire, T. Yan, L. Luo, A. Tirumala, Q. Cao, T. He, J. Stankovic, T. Abdelzaher, B. Krogh, Lightweight Detection and Classification for Wireless Sensor Networks in Realistic Environments. *In Proc. of the 3rd ACM Conf. SenSys'05*, November.
- [25] S. Mohan, F. Mueller, D. Whalley and C. Healy, Timing Analysis for Sensor Network Nodes of the Atmega Processor Family, Real-Time and Embedded Technology and Applications Symposium, March 2005.
- [26] S. J. Leon, Linear Algebra with Applications 5th Ed. Prentice Hall, NJ, 1998.
- [27] T. Canli, M. Terwilliger, A. Gupta, and A. Khokhar, Power-Time Efficient Algorithm for Computing FFT in Sensor Networks, *In Proc. of the 2nd ACM SenSys'04*, November.
- [28] A. Grama, A. Gupta, G. Karypis, and B. Kumar, Introduction to Parallel Computing, 2nd Ed. Pearson Education Limited, 2003
- [29] V. Shnayder, M. Hempstead, B. Chen, G. W. Allen, and M. Welsh, Simulating the Power Consumption of Large Scale Sensor Network Applications. *Proc. 2nd ACM SenSys'04*, Nov.
- [30] P. J. Eberlein and H. Park, Efficient Implementation of Jacobi Algorithms and Jacobi Sets on Distributed Memory Architectures. *Journal Parallel Distributed Computing* 8(4), (1990)
- [31] P. J. Eberlein, On the Schur Decomposition of a Matrix for Parallel Computation. *IEEE Transaction Computers* 36(2), 1987
- [32] TinyOS Tutorial-Lesson 4: Component Composition and Radio Communication, 2003. Last accessed May 10, 2006. www.tinyos.net/tinyos-1.x/doc/tutorial/lesson4.html
- [33] M. Srivastava, Sensor node platforms and energy issues. *Mobicom 2002 Tutorial*. Available online and last accessed May 12, 2006 nesl.ee.ucla.edu/tutorials/mobicom02/slides/Mobicom-Tutorial-2-MS.pdf
- [34] T. Canli, M. Terwilliger, A. Gupta and A. Khokhar, Power Efficient Algorithms for Computing Fast Fourier Transform over Wireless Sensor Networks. *The 4th ACS/IEEE Conf. Computer Systems and Applications*, Dubai, UAE, March 2006.