*Elektrotehniški vestnik 74(3): 107–112, 2007*
*Electrotechnical Review, Ljubljana, Slovenija*

# An implementation of a two-layered SVM classifier in Condor

**Mira Trebar**[1], **Nigel Steele**[2]

[1] *University of Ljubljana, Faculty of Computer and Information Science, Tržaška 25, Ljubljana, Slovenija*
[2] *Department of Mathematical Sciences, Coventry University, Priory Street, Coventry, CV1 5FB, UK*
*E-mail: mira.trebar@fri.uni-lj.si*

**Abstract.** Condor, as a high-throughput distributed computing system, is used in a two-layered Support Vector Machine (SVM) implementation of the classification problem. The complexity of the SVMs training algorithm increases with respect to the number of samples. The data are split into subsets and the solution described reduces the training time by optimizing the first layer SVMs separately on a cluster of computers. As a result, a smaller subset of support vectors from partial results is used to optimize the second layer SVM. For the experiments on a large data set (Forest data), the distributed implementation of two-layered SVMs in Condor shows a significant improvement of the training time by keeping or even improving the error performance of a single SVM classifier.

**Key words:** Support Vector Machine, classification, distributed computing, Condor

## Izvedba dvo-nivojskega SVM klasifikatorja v sistemu Condor

**Povzetek.** Predstavili bomo porazdeljen upravljalski sistem Condor, ki je uporabljen za dvonivojsko izvedbo modela SVM (Support Vector Machine) pri reševanju problemov klasifikacije. Kompleksnost optimizacijskega algoritma SVM zelo hitro narašča s številom učnih vzorcev. Učni vzorci so naključno razvrščeni v enako velike podmnožice, ki so uporabljene za porazdeljeno učenje prvega nivoja modela SVM v gruči računalnikov. Rezultat prvega nivoja je bistveno manjša množica učnih vzorcev, imenovanih 'support vectors', ki jo uporabimo za učenje drugega nivoja modela SVM.

S poskusi dvonivojskega modela SVM in uporabo sistema Condor smo ugotovili, da lahko za obsežne množice vzorcev (Forest data) bistveno skrajšamo čas učenja in v primerjavi z enim modelom SVM zagotovimo enako natančnost klasifikacije ali jo celo izboljšamo.

**Ključne besede:** SVM, razvrščanje, porazdeljeno računanje, Condor

## 1 Introduction

For many research projects it is common to have problems that require days or weeks of computation on personal computer to solve. There exist different solutions and one of them is to use a cluster of computers, workstations or all available resources in an organization connected by a local-area network (LAN). This can be done by using the Condor software system which enables a High Throughput Computing (HTC) environment [1] which delivers large amounts of computer power over a short period of time. Condor can save time when a job must be run many different times with hundreds of different data sets and streamlines the scientist's tasks by allowing the submission of many jobs at the same time. In this way, tremendous amounts of computation can be done with very little intervention from the user [2]. Moreover, Condor allows users to take advantage of idle machines that they would not otherwise have access to. Condor also provides other important features for its users, because the source code does not have to be modified in any way. Condor is a specialized workload management system for computer-intensive jobs developed as the product of the Condor Research Project at the University of Wisconsin-Madison and is currently available as a free download from the Internet [3].

Support Vector Machines (SVMs) are presented as a machine learning method in classification and also regression problems [9]. However, they require the solution of a quadratic optimization problem which means that with larger data sets the complex-

*Received 23 May 2006*
*Accepted 26 February 2007*

ity of SVMs increases. The original Support Vector Machine (SVM) is a linear classifier [6] in the input or data space that is mapped into a higher dimensional feature space, resulting in a nonlinear classifier. As a result, SVMs have the property of encapsulating all the information using a small number of data called support vectors. In the last few years, several attempts to classify large data sets with various SVMs models have been published: the large quadratic programming (QP) problem is broken down into a series of smaller QP sub-problems [7], a parallel mixture of SVMs [4] and the SVM$^{light}$ fast implementation algorithm [5]. All the experiments showed significant time improvement by using more expert SVMs trained on different subsets of data.

We introduce here the parallel implementation of two–layered model of SVMs. The training data set is randomly partitioned into data subsets of size $N/M$ ($N$ is the number of input patterns and $M$ is the number of subsets). The first layer consists of $M$ Support Vector Machines (SVMs) optimized in parallel and as a result we obtain smaller sets of support vectors (SVs) to use them as a second layer support vector training set. Actually, the proposed approach is presented as a two-layered model of SVMs, but only the results of the second layer SVM are relevant for the classification of new input data.

Experimental results described in the paper include traditional job execution on one machine as well as different ways of job processing using Condor's dedicated cluster. Comparisons of results made under different conditions of the cluster are given, along with the use of the Condor software.

## 2 Condor software system

Condor is a specialized software system for computer intensive jobs which effectively utilizes the workstations, dedicated clusters of workstations and personal computers that communicate over a network. As a batch system, Condor provides a job queueing mechanism, scheduling policy, priority scheme, resource monitoring, and resource management. The three powerful mechanisms used in Condor are:

- The Classified Advertisement mechanism (ClassAd) which is an extremely flexible framework for matching resource requests (jobs) with resource offers (machines).
- The Remote System Calls mechanism which assures to run any applications on any remote machine of a given architecture.
- The Checkpointing mechanism transparently records a current state of the executing program in the checkpoint file in such a way that the program can be later restarted from that state. It also permits a job to migrate from one machine to another machine.

Users submit their serial or parallel jobs to Condor and Condor places them into a queue, chooses when and where to run the jobs based upon a Classified Advertisement mechanism. While the jobs are running, Condor carefully monitors their progress, and ultimately informs the user upon completion. The use of the Remote System Calls offers an added benefit that a user submitting jobs to Condor does not need an account on the remote machine where it runs a job. When the job does a system call, for example to do an input or output function, the data is maintained on the machine where the job was submitted.

The basic system of the Condor pool in Fig. 1 is comprised of a single machine which is a central manager, and an arbitrary number of other machines which can act as submit machines or as execution machines. There is only one central manager which is very important part in the pool and it should be reliable and is likely to be online all the time. If it crashes, no further matchmaking can be performed. Conceptually, the role of Condor is to match waiting

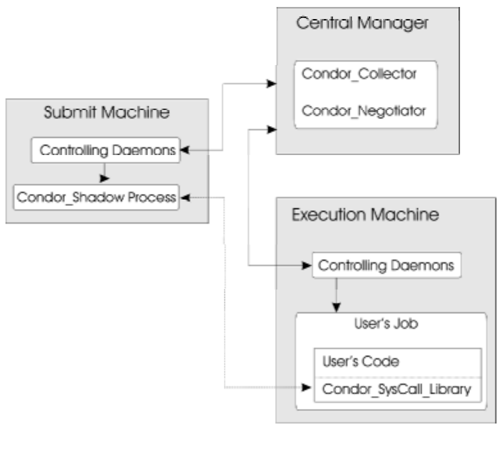

Figure 1. Condor pool

requests with available resources which is done by the match-making between available machines and submitted jobs. Jobs want to find machines upon which they can execute. The central manager acts as a collector of information and the negotiator between resources and resource requests and performs these services by two separate programs, named daemons (*Condor_Collector*, *Condor_Negotiator*). Machines have specific resources available, such as the platform and the amount of available memory. When submitting a job, a separate ClassAd is produced for each job and machine, listing all attributes. Condor acts as a matchmaker between the jobs and the machines by pairing the ClassAd of a job with the ClassAd of a machine and it will assure that the requirements in both ClassAds are satisfied.

Any machine in the pool, including the central manager, can be configured to execute Condor jobs and also as a machine from where jobs can be submitted. Here is important to mention that the resource requirements for a submit machine are actually much greater than the resource requirements for an execute machine. Both machines perform their services with controlling programs (*Controlling Daemons*) when they are communicating with the central manager. The execution machine running the users' job also performs the system call (*Condor_SysCall_Library*) which is sent back to the submit machine where it is managed by *Condor_Shadow Process* running whenever the job was submitted. A submitted job may also prefer to execute on a machine with better floating point facilities, or it may prefer to execute on a specific set of machines. These preferences are also expressed in the ClassAd. Further, a machine owner has great control over which jobs are executed under what circumstances on the machine. The owner writes a configuration file that specifies both requirements and preferences for the jobs. Alternatively, any of these may be expressed as a preference, for example where the machine prefers the jobs of a select group, but will accept the jobs of others if there are no jobs from the select group.

To illustrate simply how Condor works, we present here four steps needed to prepare and run the job:

- Code preparation includes an application that runs in the background which is not able to do the interactive input and output.
- Select the runtime environment, also called Condor universe (standard, Vanilla, ...).
- Write submit description file with all the information about the job running, the files to use, how many times to run the job.
- Submit the job with the *condor_submit* command which takes as an argument the name of the file called a submit description file.

For programs, running in a sequential and parallel way, where the input, output or execution of one or more programs is dependent on one or more programs, a directed acyclic graph (DAG) is used. The programs are nodes in the graph and are linked-up by the parallel or serial dependency. The Directed Acyclic Graph Manager (DAGMan) is a meta-scheduler and submits jobs to Condor in an order defined by DAG and returns the result. The DAG is described in an input file which includes all Condor submit description files with their execution dependencies and parameters. Each node in a DAG is a unique executable file and each has a unique Condor submit description file. The jobs used in DAG are submitted using the program *condor_submit_dag* which takes as an argument the name of the input file.

## 3 Two-layered Support Vector Machine

Support Vector Machines (SVMs) [9], [6] have been applied to many classification problems, generally yielding good performance and acceptable execution times for smaller data sets. The original SVM is a linear classifier where the input vectors are separated by a hyperplane, while the non–linear SVMs map the input vectors from the original space into a high–dimensional feature space, where they become linearly separable and there construct an optimal hyperplane [9].

The output function of nonlinear SVMs for classification problem with training data given as $\mathbf{x} \in \Re^n$, $y \in \{+1, -1\}$ is then defined as

$$y = \sum_{i=1}^{N} \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i) + b \tag{1}$$

where $K(\mathbf{x}, \mathbf{x}_i)$ is a kernel function, $\mathbf{x}$ is the input vector of a test example, $y_i$, is a class label and $\mathbf{x}_i$ is the input vector for the $i$–th training example, $N$ is the number of training samples, $\boldsymbol{\alpha} = \{\alpha_1, \dots, \alpha_N\}$ and $b$ are the parameters of the model. To find the coefficients $\alpha_i$, $i = 1, \dots, N$, it is sufficient to solve the optimization problem in the dual space by finding the minimum of the objective function

$$Q(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i y_i \alpha_j y_j K(\mathbf{x}_i, \mathbf{x}_j). \tag{2}$$

Because training vectors will often repeat, the soft margin optimal hyperplane is usually determined with the upper bound $U$ on the parameters $\alpha_i$, and the optimization from (2) is computed as subject to the constraints

$$\sum_{i=1}^{N} \alpha_i y_i = 0 \quad \text{and} \quad 0 \leq \alpha_i \leq U \ , i = 1, \dots, N.$$

Actually, the performance of nonlinear SVMs depends on the kernel function. By the use of a kernel function, the mapping operation and all calculations associated with it are actually carried out in data space. Very often a Gaussian Radial Basis function (RBF) defined as

$$K(\mathbf{x}, \mathbf{x}_i) = \exp\left\{-\frac{||\mathbf{x} - \mathbf{x}_i||^2}{\sigma^2}\right\} \tag{3}$$

with common variance $\sigma$ is used.

Further, we introduce a two-layered organization of SVMs shown in Fig. 2. The first, parallel layer of SVMs is used only in the training phase while the second layer SVM is used for finding the final decision hyperplane and output function that satisfy the training data set. A set of input samples or training data are given as $\mathbf{x} \in \Re^n$, $y \in \{+1, -1\}$, and is partitioned into $M$ training subsets $\mathbf{x}_s(k) \in \Re^n$, $y_s(k) \in \{+1, -1\}$, $k = 1, 2, \ldots, M$. The number of subsets $M = 2, 3, \ldots$ should depend on the size of data set and is experimentally determined. In the two-class classification problem, the training data set consists of $N_1$ input samples from class $C_1$, and $N_2$ input samples from class $C_2$. Samples are selected at random and placed into $M$ training subsets ($\mathbf{x}_s(k)$), $y_s(k)$, $k = 1, 2 \ldots M$. The number of training samples in each subset is chosen as $\dfrac{N_1}{M} + \dfrac{N_2}{M}$.

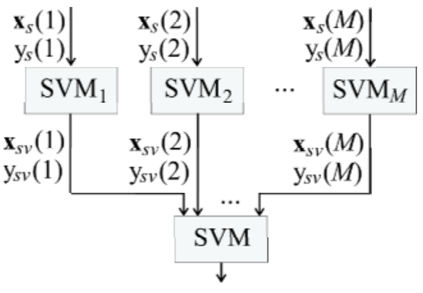

Figure 2. Two-layered SVM model

We now describe our training approach as a two-layered training algorithm. The first layer includes parallel training of SVMs over all $M$ subsets. After training the first layer SVMs with subsets $\mathbf{x}_s(k)$, $k = 1, 2, \ldots, M$ we get $M$ sets of support vectors $\mathbf{x}_{sv}(1)$, $\mathbf{x}_{sv}(2)$, $\ldots$, $\mathbf{x}_{sv}(M)$ with a relatively small number of input samples in comparison to whole data set of $N$ input samples and they form a set of support vectors $\mathbf{x}(1) = \mathbf{x}_{sv}(i)$ with outputs $y(1) = y_{sv}(i)$, $i = 1, 2, \ldots, M$ used as training set for the second layer. Finally, we get a new set of support vectors with learning parameters and we use the second layer SVM as a single classifier to evaluate training set and testing set.

## 4 Experimental work

To show a general use of the proposed two-layered SVMs and to run the program as a distributed computing with Condor software system, we performed the experiments on part of the UCI Forest data set [8]. The classification problem of 7 classes was modified in a binary classification problem where class 2 was separated from the other 6 classes. From more than 500000 samples with 54 attributes, we prepared several data sets which range from 16000 (16T) samples to 512000 (512T) samples. For each data set the input samples were divided into two sets: the training set (90%) and the test set (10%). Finally, the training sets were divided into subsets with the number of subsets $M = 2, 4, 8, 16, 32$. We performed the experiments on six data sets (16T, 32T, 64T, 128T, 256T and 512T) obtained from the described Forest data. For each training data set we prepared the distributed application with different number of subsets ($M = 2, 4, 8, 16, 32$).

All experiments for single SVM were made directly with SVM$^{light}$ implementation of Support Vector Machines [5]. The same SVM$^{light}$ implementation was used in two-layered SVM model where it was arranged in a hierarchical mode. First layer was defined with $M$ SVM$^{light}$ implementations where the support vectors were generated during the training phase. Thereafter, they were combined in a new training set of support vectors which was used on the second layer SVM$^{light}$ implementation. The accuracy of the proposed implementation was measured by using only the second layer SVM$^{light}$ optimization parameters.

Since SVM$^{light}$ implementations with a radial basis function was used, Gaussian kernels with variance ($\sigma$=4) and regularizing parameter ($U$=5), were obtained by 10–fold cross validation on two additional data sets with 5000 and 20000 samples. We kept here a test set of 10% examples in data set to compare the best two-layered SVM model to the single SVM and we also had a validation set of 10% of examples from the training data set to define the parameters used in experiments.

To run the two-layered SVM classifier, we prepared the DAG applications for the Condor pool. It consisted of several submit machines and of a cluster of 16 machines. One of the machines in the cluster is the Central Manager and runs using the Linux operating system, while the other 15 machines are used as execution machines, running on Windows XP operating system. All computers in the cluster are Intel Pentium 4, 3GHz machines with 1GB memory and 80 GB discs. The submit machines in our Condor pool are personal computers connected in the local area network.

### 4.1 Distributed DAGMan applications

The DAGMan application consists of an input file which describes the directed acyclic graph (DAG), and several submit description files for each program which represents the execution of the node in the DAG. Fig. 3 shows the directed acyclic graph for training two-layered SVM model with M=4. For the

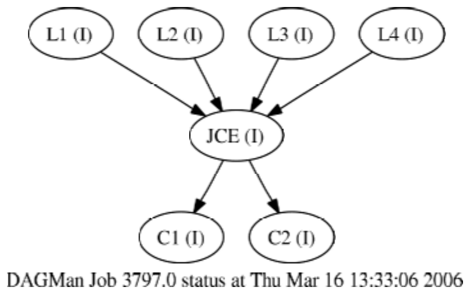

Figure 3. Directed acyclic graph (DAG) defined for $M = 4$

presented DAG (data set = 32T, M=4), the input file specifies at the beginning all jobs for the training of subsets performed in parallel.

```
job L1 learn.sub
job L2 learn.sub
job L3 learn.sub
job L4 learn.sub
```

In each job, the variables define the name of the training set (class), the number of subsets (nsets) and the index of subset used in particular job (n=‘i’), $i = 1, 2, 3, 4$ are defined.

```
vars L1 class='forest32t' nsets = '4' n='1'
vars L2 class='forest32t' nsets = '4' n='2'
vars L3 class='forest32t' nsets = '4' n='3'
vars L4 class='forest32t' nsets = '4' n='4'
```

The second job JCE performs the second layer SVM training, where all subsets of first layer support vector (SV) are used as a training data. The variables used are called class and nsets.

```
job JCE learn1.sub
vars JCE class='forest32t' nsets = '4'
```

When the second layer training is finished, the model performance is obtained by classifying the training and test sets with two jobs C1 and C2 which are executed in parallel.

```
job C1 classifysvmtrain.sub
job C2 classifysvmtest.sub
vars C1 class='forest32t'
vars C2 class='forest32t'
```

At the end, we define the dependencies in the DAG by defining jobs as parent and as child.

```
parent L1 L2 L3 L4 child JCE
parent JCE child C1 C2.
```

Each job which is executed by DAGMan has its own submit file. In the submit file we define:

- the universe used is vanilla with the File Transfer mechanism on Windows platforms,
- execution machine requirements with the operating system (opsys == 'WINNT51') and the name of the machine in Condor pool (machine == 'slave2.cluster1.laspp'),
- executable, input and output data files which refer to the program in the DAG nodes,
- if Condor transfers output files and when to transfer them,
- names of error and log files which include all the information about the job execution.

We have the same submit and execution files in all DAGMan applications and only the input files describing the DAG differ due to the names of the input files and the number of subsets.

## 4.2 Classification performance and training time

Now, let us first show the results of two-layered SVM classifier in comparison to single SVM classifier.

Table 1. Classification results ($M$=4)

| Data | Single SVM | | Two-layered SVM | |
|---|---|---|---|---|
| | Training | Testing | Training | Testing |
| 16T | 98.46 | 92.18 | 98.53 | 91.93 |
| 32T | 98.60 | 95.19 | 98.51 | 95.03 |
| 64T | 98.73 | 96.95 | 98.74 | 96.91 |
| 128T | 98.07 | 96.91 | 98.17 | 96.99 |
| 256T | 97.38 | 96.26 | 97.58 | 96.39 |
| 512T | 96.70 | 96.10 | 96.95 | 96.27 |

In case of two-layered SVM model we obtained very similar results for all experiments with different numbers of subsets. In Table 1 we present only the best results for $M$=4. As can be seen, for larger data sets, the classification results of two-layered SVM model are slightly better than the classification results of a single SVM.

Table 2. Execution time in seconds

| $M$ | Data | | | | | |
|---|---|---|---|---|---|---|
| | 16T | 32T | 64T | 128T | 256T | 512T |
| 1 | 195 | 604 | 1455 | 6359 | 29865 | 153362 |
| 2 | 164 | 483 | 926 | 4401 | 18488 | 92002 |
| 4 | 442 | 360 | 819 | 3144 | 14063 | 74547 |
| 8 | 675 | 503 | 782 | 3461 | 15349 | 77870 |
| 16 | 969 | 631 | 1142 | 3762 | 15720 | 83986 |
| 32 | 1234 | 1029 | 1743 | 4092 | 17512 | 93958 |

The training time as a function of the number of subsets ($M$) which is obtained by proposed DAGMan applications presented in Table 2 shows that the two-layered SVM model outperformed a single SVM ($M$=1) in the case of all data sets. The improvement for larger data sets is more evident in case of the number of subsets $M$=4.

In Fig. 4 we can see the comparison of the training time based on the number of subsets ($M$) on three different data sets. In the case of the first data set (16T) there is a small improvement in the training time for $M$=2 and there is an improvement on the other two data sets (32T, 64T) for $M$=4. For all other data sets with a larger number of training samples (128T, 256T, 512T) the best training time results are also obtained using $M$=4.

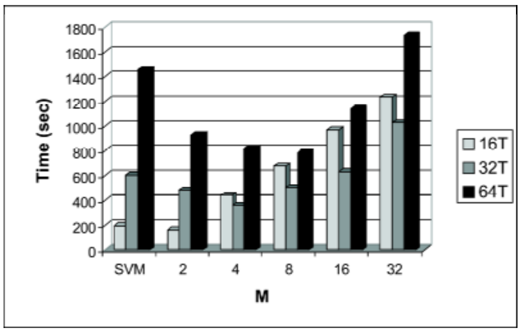

Figure 4. Training time (16T, 32T, 64T)

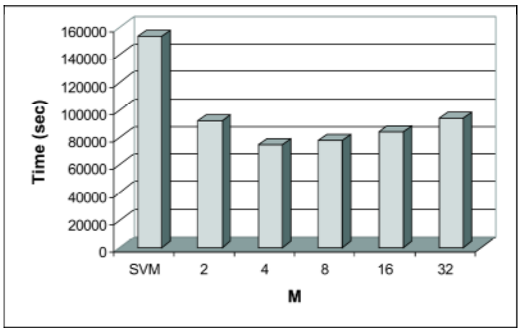

Figure 5. Training time (512T)

With larger numbers of subsets ($M$) the execution time starts to increase faster for small data sets in Fig. 4 and more slowly for larger data sets (512T) which is shown in Fig. 5.

## 5 Conclusion

In this paper, we presented a parallel implementation of the Support Vector Machine as a two–layered classification model. The performance of the proposed model trained with a large data set on several numbers of subsets is comparable with the single SVM model trained on the whole training set, but is significantly better in terms of the training time. Extensive comparisons suggest that we can define training parameters by 10-fold cross-validation on smaller data sets and use them on larger data sets. The number of subsets has an important impact on the training time. If the number of subsets is small, the number of support vectors will be relatively small and the second layer training time is small. The opposite occurs when there is a large number of subsets where we have a larger number of support vectors which increases the second layer training time. In our experiments, the best selection is four subsets in terms of accuracy and also training time.

We can conclude that the use of the proposed two-layered SVM, based on data set partition into data subsets, is a very promising method when applied to larger data sets because we have comparable performance results and the first layer SVMs can run in a distributed computer system.

## 6 References

[1] D. Thain, T. Tannenbaum, M. Livny, Distributed Computing in Practice: The Condor Experience, *Concurrency and Computation: Practice and Experience*, Vol. 17, No. 2-4, pages 323-356, February-April, 2005.

[2] T. Tannenbaum, D. Wright, K. Miller, M. Livny, *Condor - A Distributed Job Scheduler, in Thomas Sterling, Beowulf Cluster Computing with Linux*, The MIT Press, 2002.

[3] Condor Manual, http://www.cs.wisc.edu/condor, 2004.

[4] R. Collobert, S. Bengio, Y. Bengio, A parallel mixture of SVMs for very large scale problem, *Neural Computation*, 14(15), 2002.

[5] T. Joachims, SVM*light* Support Vector Machine, http://www.cs.cornell.edu/People/tj/svm_light,V6, 2005.

[6] V. Kecman, *Learning and Soft Computing*, London, MIT Press, 2001.

[7] E. Osuna, R. Freund, F. Girosi, An Improved Training Algorithm for Support Vector Machines. *Neural Networks for Signal Processing VII*, Proceedings of the 1997 IEEE Workshop, pages 276-285, Ney York, 1997.

[8] UCI KDD archive, http://kdd.ics.uci.edu/databases /covertype/covertype.html, 2005.

[9] V. N. Vapnik, *The Nature of Statistical Learning Theory*, New York, Springer-Verlag, 2000.

**Mira Trebar** received her M.Sc. and Ph.D. degrees in electrical and computer engineering from the University of Ljubljana, in 1992 and 1997, respectively. She works as a lecturer at the Faculty of Computer and Information Science in Ljubljana. Her research interests are in learning methods, soft computing and in distributed computing.

**Nigel Steele** is Emeritus Professor of Mathematics in Coventry University. He has research interests in neural networks and fuzzy systems. He is Honary Secretary of the Institute of Mathematics and its Applications, with responsibility for educational matters.