Proceedings of the 2009 IEEE International Conference on Systems, Man, and Cybernetics
San Antonio, TX, USA - October 2009

A GA-SVM Feature Selection Model Based on High Performance Computing Techniques

Tianyou Zhang*, Xiuju Fu*, Rick Siow Mong Goh*, Chee Keong Kwoh^, Gary Kee Khoon Lee*
*Institute of High Performance Computing, 1 Fusionopolis Way, #16-16 Connexis, Singapore 138632
^Nanyang Technological University, Singapore 637457
Contact Email: zhangty@ihpc.a-star.edu.sg

***Abstract*—Supervised learning is well-known and widely applied in many domains including bioinformatics, cheminformatics and financial forecasting. However, the interference from irrelevant features may lead to the poor accuracy of classifiers. As a popular feature selection model, GA-SVM is desirable in many of those cases to filter out irrelevant features and improve the learning performance subsequently. However, the high computational cost strongly discourages the application of GA-SVM in large-scale datasets. In this paper, an HPC-enabled GA-SVM (HGA-SVM) is proposed by integrating data parallelization, multithreading and heuristic techniques with the ultimate goal of robustness and low computational cost. Our proposed model is comprised of four improvement strategies: 1) GA Parallelization, 2) SVM Parallelization, 3) Neighbor Search and 4) Evaluation Caching. All the four strategies improve various aspects of the feature selection model and contribute collectively towards higher computational throughput.**

***Keywords*—genetic algorithm, support vector machine, HPC**

I. INTRODUCTION

Selecting important features out of the original feature set is a challenging task. Given data samples with class labels, supervised classification models are usually used together with optimization algorithms for feature selection in which classification accuracies are used as fitness evaluation of the selected feature subsets. In this work, we develop a feature selection model which combines the merits of support vector machine (SVM), genetic algorithm (GA) and high performance computing techniques.

Support vector machine [1] is one of the most popular classifiers used in supervised learning. The principle of SVM is constructing an optimal separating hyperplane to maximize the margin between two classes of data. The performance of SVM classifier is often sensitive to the choice of margin cost C and kernel parameters [2]. The optimal parameters that lead to the minimal generalization error are data-dependent. Presently no rules or formula can be used to compute such values analytically, so parameter tuning is often required. An intuitive realization of parameter tuning is grid search [3]. That is, the parameters are varied by step-size within the preset range of values (in a structure of "grid"); the optimal values can be found by measuring every combination (every node in the grid). Due to its complexity, usually two-dimensional grid is used to tune a pair of parameters such as C and γ (Gaussian function width in RBF kernel).

Even after parameter tuning, SVM classifier might deliver poor accuracy in classifying some particular datasets. One possible reason is noise interference in which an overwhelming number of irrelevant features are included inside the inputs so that a truly representative classifier cannot be learnt. If prior knowledge is insufficient to differentiate which features are truly relevant to the output, such that all the possible features are included in the training data, the accuracy of learning would be deteriorated. In those cases, the key of improving learning performance is feature selection, which is the technique of selecting only the relevant features to build a robust learning function.

There are many optimization techniques that have been used with supervised learning in feature selection. One of the favored choices is Genetic Algorithm (GA) [4]. GA is a search technique that is inspired by the natural evolution. In the evolution, the individuals with better genetic merits (chromosome) are more likely to survive under natural selection and reproduce the offspring; the unfit ones are filtered out. By constant filtering, generation after generation, the population tends to carry the fitter and fitter chromosomes. To mimic this process, all candidate solutions to a feature-selection problem can be encoded as "chromosome" (feature subset representation), which takes the form of bit array (e.g. 1001…0101) — 1s and 0s denote the presence or absence of each feature. A group of those candidate solutions is sampled randomly and form the initial population of chromosomes. The chromosomes are then evaluated by objective function to compute the fitness scores. Multiple chromosomes are stochastically selected based on their fitness, recombined (crossover) and mutated, and finally form the next generation. By means of random mutation and crossover, the variety of chromosomes is introduced and evaluated in every generation and gradually evolve the solutions towards the optimal. The process will be iterated until convergence, i.e. there is no more improvement to the best fitness score in the population. At the end, the chromosome with the best-ever fitness will be the final solution and all the features denoted by 0s will then be filtered out.

GA-SVM had been widely used to filter out the irrelevant features and improve the learning accuracy in the noisy settings. However, one practical problem of GA-SVM is its extremely high computational cost. Assuming m population and g generations in GA, w x w grid in parameter tuning and t seconds for SVM plus 10-fold cross validation, then the overall runtime will cost mgw^2t seconds. It is a time-consuming process that even a small-scale problem may need nearly a day to complete (demonstrated in the Result section). That strongly discourages the application of GA-SVM to larger and more

978-1-4244-2794-9/09/$25.00 ©2009 IEEE

complex data. In this paper, we introduce high performance computing (HPC) techniques and heuristic methods to speed up the traditional GA-SVM feature selection model. In our HPC-enabled GA-SVM (HGA-SVM), we employ data parallelization, multithreading, repeated evaluation reduction and heuristic optimization, with the ultimate goal of trimming down computational cost and making large-scale feature selection more feasible. The HGA-SVM is comprised of four improvement strategies: 1) GA Parallelization, 2) SVM Parallelization, 3) Neighbor Search, and 4) Evaluation Caching. All the four strategies work collectively towards higher computational efficiency.

II. METHODOLOGY

A standard GA-SVM feature selection model is comprised of three operators: crossover, mutation, and SVM evaluation. With respect to population size, the first two are of linear complexity; and the last one is of quadratic complexity. It is clear that reducing population size could lower computational cost effectively. Moreover, when input data grows larger and more complex, most of time lag would rise in SVM evaluation since all other operators only work in chromosome layer. Imagine if a single SVM training is slowed down by *t* seconds in a larger dataset, by the effect of 10-fold cross validation and 10x10 grid search, the time lag of each chromosome in every generation would be amplified to 1000t seconds. Apparently SVM evaluation is the biggest factor in GA-SVM that discourages the application in large-scale dataset and so speedup specific to SVM would be desirable.

Identical chromosomes may re-emerge in the different GA generations due to random mutation and crossover. Since a standard GA does not keep any historical records on the past-evaluated chromosomes, it has to evaluate every appearance of those identical chromosomes. That will waste the computation power for the unnecessary evaluations and increase the runtime.

Exhaustive grid search for parameter tuning is slow despite the grid dimension is small. In a simple 10x10 grid search, it requires 100 times of SVM learning (with 10-fold cross validation) for each chromosome in every generation. This exhaustive search will incur a huge computational cost. However, parameter tuning cannot be omitted even though the cost is high. Otherwise SVM learning would be biased and the purpose of improving learning performance is undermined.

Four improvement strategies were designed to alleviate computational cost. Parallel GA and parallel SVM speed up the GA and SVM respectively; neighbor search replaces grid search to reduce the number of combinations to be measured; and lastly evaluation caching avoids the repeated unnecessary evaluations. Figure.1 shows the workflow of all four improvement strategies in HGA-SVM.

A. Parallel/Distributed GA

The design of GA parallelization follows parallel island model [5] in a coarse grained architecture. The entire population of chromosomes is divided into *n* subpopulation and each subpopulation is assigned to a different parallel node. Every parallel node evolves their local subpopulation by a serial GA. At the end of every generation, multiple chromosomes are selected randomly at each node and exchanged among the peers, which is called "migration". Migration brings in the new variety to local population and facilitates to build up the common trend of evolution in all subpopulations.

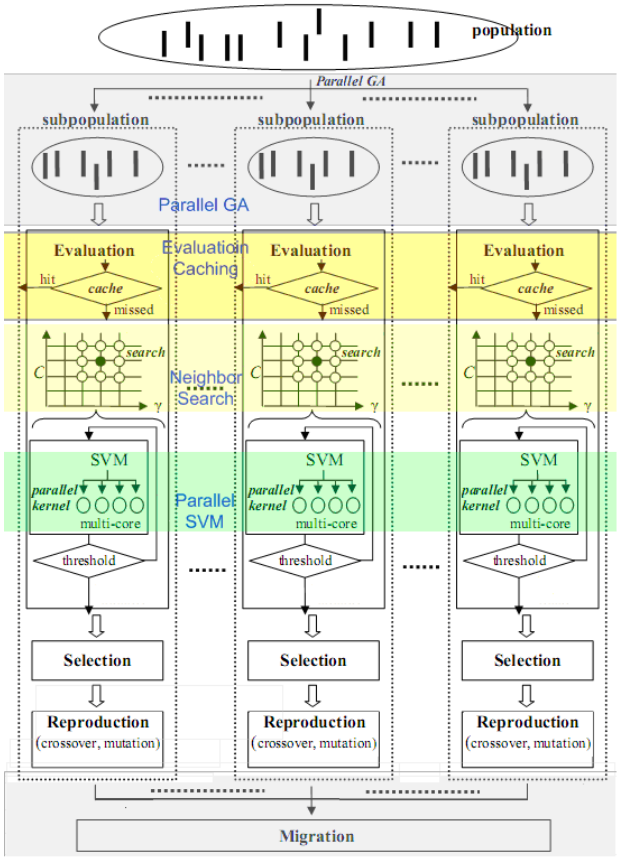

Figure 1. Design Scheme of HPC-enabled GA-SVM feature selection model

By distributing chromosomes to *n* parallel nodes, local population size is reduced by factor of $1/n$ (population size is an even integer before and after reduction). The execution time of a single GA generation is speeded up by *n* times approximately, because selection, crossover and mutation are of $O(n)$ complexity and SVM evaluation is of $O(n^3)$ complexity with respect to population size [6]. There are however some drawbacks of parallelization — parallel overheads, which includes start-up/termination overhead, synchronization overhead and communication overhead. The first two overheads are unavoidable in order to coordinate parallel computing in multiple nodes; so we focus on reducing communication overhead in this study. As adoption of parallel island framework, GAs run independently at different nodes with their local copy of data, so the necessity of data communication is minimized. Another reduction is realized in migration operation, which requires passing around the arrays of bits (chromosomes) among the nodes. There are many migration schemes specific to certain topology in the literature. To minimize communication overhead, we adopt "ring"

topology for migration in which each node transfers the local best chromosomes to its neighbor on the ring. For instance, among three parallel nodes *A*, *B* and *C*, the exchange will happen as $A \rightarrow B$, $B \rightarrow C$ and $C \rightarrow A$. Then the incurred communication overhead is linear w.r.t. the number of parallel nodes. If there are *n* parallel nodes, it requires *n* generations for the migrated chromosomes to travel in the ring and return to their original node. Therefore any serial GA should be allowed for termination only if there is no further improvement for at least *n* consecutive generations. Once any serial GA terminates, the parallel GA will stop the iteration and collect the local best chromosome from individual nodes to compute the final solution.

GA parallelization is developed using the MPI library [7] and is usually catered for the distributed-memory hardware architecture. Parallel GA using MPI is able to scale to all the compute nodes available, which would significantly benefit the application of our HGA-SVM in large-scale datasets. With the introduction of multicore processors, GA parallelization can also be applied to mainstream shared-memory systems to achieve good performance speedup as well.

B. Parallel SVM

SVM training is compute-intensive because it requires quadratic programming (QP) [1] for determining the optimal separating hyperplane. Over the years, several methods had been developed to lower the computation cost. One of the methods is sequential minimal optimization (SMO) [8]. SMO divides a large QP problem into a series of smaller QP problems that can be solved analytically such that the training is speeded up. However, when SVM is required to repeat by thousands of times in a typical feature selection task, the SVM with SMO (SVM-SMO) is still considerably slow.

To speedup the SVM-SMO, we applied the parallelization technique to distribute the computations to multiple nodes/threads for concurrent execution. As parallelization introduces the extra overheads in coordination and communication, it is wise to parallelize the most computational intensive section to achieve the maximal speedup. Table I shows a typical execution profile of SVM-SMO (retrieved from LibSVM [9] training execution). It is clear that kernel calculations take up most of the computational time.

TABLE I. A TYPICAL EXECUTION PROFILING FOR SVM-SMO (LIBSVM)

Time (%)	Self (sec)	Calls (sec)	Function Name
81.15	231.34	791,262,338	Kernel::kernel
11.67	33.27	269,460	SVC_Q::get_Q
4.78	13.64	66,095	Solver::select_working_set
2.22	6.34	1	Solver::Solve
0.15	0.44	66	Solver::do_shrinking
0.02	0.05	3,806	Cache::swap_index

The caller function of those kernel calculations is comprised of an iterative loop scanning through the instance space to select a pair for optimization. Thus OpenMP [10], a parallelization protocol designed for shared-memory multi-processor/multi-core systems, is most suitable to apply. The implementation of the parallel SVM is relatively simpler than GA parallelization. It could be done by identifying and resolving data dependency inside the loop, followed by inserting the OpenMP directives, without any modification to the structure of the algorithm. The parallel SVM will be able to utilize the multiple CPU cores concurrently in form of multithreading (refer to Figure 1), so effectively reduce the computation time. Since OpenMP also introduces the overheads, the parallel SVM would perform more efficiently if the training dataset is sufficiently large.

In our HGA-SVM, the GA and SVM operations are parallelized using MPI and OpenMP respectively. The parallelization techniques used in both of these operations allow them to work together as hybrid parallelization to speed up the workflow concurrently.

C. Neighbor Search

Parameter tuning is crucial to achieve minimal generalization error in SVM learning; however it is also time-consuming when employing exhaustive grid search. Inspired by pattern search method [11], we proposed a new derivative-free method, neighbor search, as a general solution to parameter selection problem. Neighbor search inherits the underlying structure from grid search but not attempt to measure every node in the grid. In our context, the parameters to be tuned are margin cost *C* and RBF kernel width γ. The neighbor search for *C* and γ starts from an initial position in the grid (says 10x10 grid) as the centroid and sample multiple neighbor nodes with uniform distribution within the grid of parameter domains. The centroid and its neighbors are measured by SVM learning accuracy with the corresponding pairs of parameters applied, and the best node (the one associated with the highest accuracy) is nominated as the new centroid. By repeating the above process, the centroid will keep moving towards the best node until the convergence, i.e. the centroid itself is the best among the group of examinees. Neighbor search is a heuristic search method and the confidence level of its solution depends on sampling size, i.e. how many neighbor nodes are sampled in every round. If a larger sampling size, the solution is more likely to be the optimal in the grid but slower as more measurements to be done; if a smaller sampling size, less confident to the solution but faster. By introducing neighbor search, the tradeoff between solution confidence and runtime cost could be adjusted appropriately to achieve considerable speedup with the bearable suboptimal solution.

It is intuitive that if two chromosomes differ in few bits, their optimal locations in the grid might be closer to each other. It could be applicable to the mutated chromosomes in the new generation if the hamming distance between parent and child chromosome is small. Since neighbor search has been done for the parent chromosome and found the optimal node, the same node can be used as the initial centroid for the child chromosome, which would be advantageous for the faster convergence.

D. Evaluation Caching

Evaluation caching avoids the repeated unnecessary evaluations in the different generations by building up a cache to store all the past evaluated chromosomes. Whenever an evaluation is requested, the cache is first sought. Only if a

cache-miss occurs, an SVM evaluation is executed and the cache is updated subsequently. The efficiency of the cache depends on how frequent the identical chromosomes re-emerge, which is varied and stochastic by nature. However, according to probability theory, the cache tends to be less effective when the data dimension grows, because the chance of encountering the identical chromosomes (after random mutation and crossover) decreases. The implementation of evaluation caching requires the additional memory space to store cache entries. Keeping a small footprint for the cache is a challenge as a large number of entries could be expectable. In HGA-SVM, encoding compression is introduced to reduce the length of individual cache entries. There are two encoding schemes developed for different type of data. For the low dimensional dataset, a simple multi-bit encoding is used to compress a chromosome into a multi-bit symbol string, in which the compression rate depends on the number of distinct symbols to be used; for the high dimensional sparse dataset, further compression could be achieved by encoding the difference in the consecutive bits of a chromosome followed by compression to the consecutive 0s in the encoded string. The encoding compress schemes could not only reduce the footprint of evaluation cache, but also cut down the computational cost of cache search due to shorter length of every cache entry.

III. EXPERIMENTS AND RESULTS

Our HGA-SVM has been developed based on an open-source GA toolbox (GAOT) [12]. The source codes were ported to Octave [13] and incorporated with the improvement strategies including parallel GA, neighbor search and evaluation cache. The MPI required in the parallel GA is supported by MPITB library [14]. The encoding compression required in evaluation cache was coded as C++ libraries and linked to Octave for efficiency purpose. The source code of LibSVM 2.8.6 [9] was modified to implement the parallel SVM and also ported to Octave as external library which allows direct access to runtime variables in the memory. The experiment platform used is a 2x Intel Xeon Quad-core (3.0GHz) machine with 32GB memory.

The parameter setting of HGA-SVM is listed in Table II. The population/subpopulation size and crossover/mutation/migration rate are self-explanatory, and their values were set based on experience. Max-generation refers to the maximum number of generations to be evolved; and max-convergence denotes the number of consecutive generations to be waited before termination if there is no further improvement to the fitness score, which is measured by average accuracy in stratified 10-fold cross validation of SVM classifier with the tuned parameters. The minimal update level of fitness score is 0.01%. RBF kernel was used in SVM. C and γ were tuned in range of 10^{-2} to 10^{3} by a 10x10 grid with sample size of 8.

TABLE II. LIST OF PRESET PARAMETERS IN GA-SVM

GA Parameters	
# of parallel node	n
population size	80
subpopulation size	80/n
cross-over rate	60%
mutation rate	5%
migration rate	50%
max-generation	100
max-convergence	n
fitness epsilon	0.01%
SVM Parameters	
SVM kernel	RBF
cross-validation	10-fold stratified
C and γ range	10^{-2} to 10^{3}
grid size	10x10
neighbor sampling size	8

TABLE III. LIST OF DATASETS TESTED WITH GA-SVM

Name	**# of Examples**	**# of Features**	**# of Classes**
Austrian	690	14	2
Adult	1605	123	2

The dataset used in the experiment are listed in Table III. Both datasets are the pre-processed data from the LibSVM (http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/). The performance of improvement strategies was measured by computation reduction (including search reduction and evaluation reduction) and runtime reduction per generation. Search reduction refers to how much percentage of search tasks is saved compared to grid search; and evaluation reduction refers to how much percentage of evaluation tasks is saved compared to GA without cache. As GA is a stochastic process, the number of generations for convergence may vary for different runs. Thus the overall runtime is not appropriate for benchmark purpose. Instead, runtime per generation could be used to illustrate the effectiveness of the improvement strategies.

GA-SVM is time-consuming even for a small-scale dataset like Austrian. For 690 examples of 14 features, it took 1048 minutes (~17.5 hours) to complete the 16 generations of GA-SVM. The slowness affirmed our determination to speed up GA-SVM for any feasible application in practice.

Parallel GA can significantly speed up the traditional GA by distributing chromosomes to different parallel nodes. The reduction on the population size would cut down the computational cost in all GA operations especially SVM evaluation. Figure 2 confirms this expectation in the experiment with Austrian dataset. It was observed that the runtime per generation decreased when the number of parallel nodes increased. By taking average runtime into account, Figure 3 plotted the relationship between (average) runtime per generation and the number of parallel nodes. The respective speedup gains for 2, 4 and 8 nodes were 2.01, 4.00 and 8.46, which demonstrated a linear fashion in runtime reduction.

In **Parallel-SVM**, the amount of kernel computations is evenly distributed among multiple threads, and each thread is allocated to a processing core for concurrent executions. Figure 4 shows how SVM training time changes with the growing number of threads (cores) with Adult dataset. The speedups for

2, 3 and 4 threads were 1.89, 2.60 and 2.88 (equivalent to 0.94, 0.86 and 0.71 per thread) respectively, and the plot exhibited an inverse exponential fashion. That was as the result of parallel overheads. In our design, communication overhead had been minimized by data localization and faster migration algorithm. The rest of overheads (like thread start-up and termination) have a fixed cost and are independent of data size. Therefore, a better performance could be expected in dealing with larger datasets, since the cost of the overheads would be amortized.

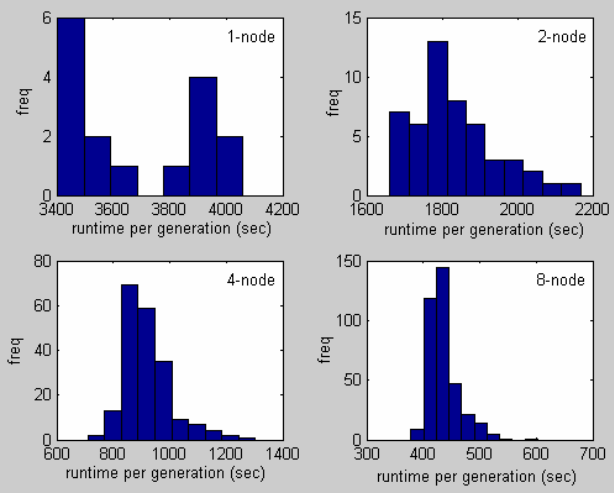

Figure 2. Distributions of Runtime Per Generation w.r.t. MPI Nodes

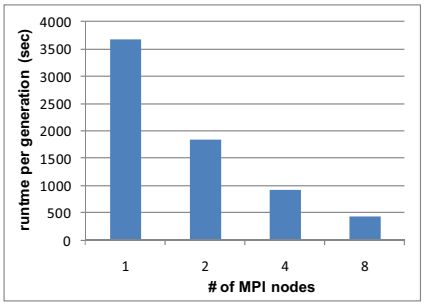

Figure 3. Parallel-GA average runtime per generation w.r.t. MPI Nodes

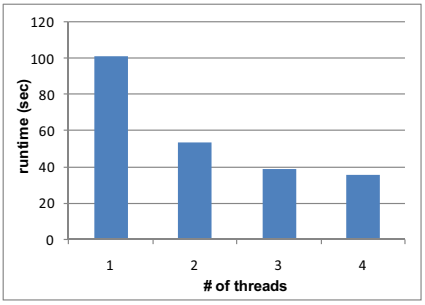

Figure 4. Parallel-SVM training time w.r.t. number of threads

Evaluation caching is able to avoid the unnecessary evaluations of identical chromosomes in the different generations. If the chance of cache hit (i.e. a chromosome has been evaluated earlier and stored in the cache) is significant, the overall speedup would be remarkable. Figure 5 shows how much percentage of evaluations was saved by caching in the experiment. The frequent re-emergence of identical chromosomes was observed as result of low feature dimension and mutation rate. 76.25% of cache hit was observed in the experiment and led to 75.62% reduction on the average runtime per generation (from 61.42 to 14.97 minutes, ~4 times speedup). The performance of evaluation caching highly depends on re-emergence probability of identical chromosomes, which is mostly affected by feature dimension. As binary encoding of GA's chromosomes, the total number of combinations of features is 2^n where n is the feature dimension. When feature dimension increases, the chance of hitting a chromosome in the past evaluations will drop rapidly. This phenomenon was confirmed in the experiment with Adult dataset. As feature dimension rose from 14 to 123 with the same population size, there were only 3 cache hits during 40 generations of GA. By considering the overhead incurred in caching, the results suggested that this strategy should be cautious in applying to high dimensional data.

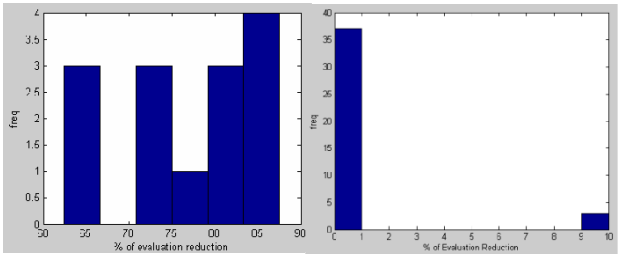

Figure 5. Evaluation Reduction Distribution for Evaluation Caching (left: Austrian, right: Adult)

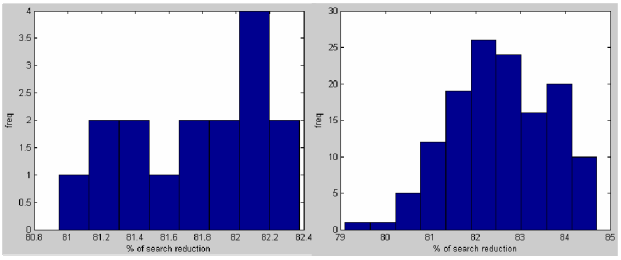

Figure 6. Search Reduction Distribution for Neighbor Search (left: Austrian, right: Adult)

Neighbor search is an improvement to grid search for tuning C and γ by replacing exhaustive search with neighbor sampling and heuristic search. Figure 6 summarizes the search reduction of neighbor search in the experiments with Austrian and Adult dataset. For Austrian dataset, two independent runs of HGA-SVM were conducted with grid search and neighbor search. Using the same 10x10 grid and parameter range, grid search required 8000 times (80 chromosomes x 10 x 10 grid) of SVM measurements per generation; and neighbor search measured only 1410 to 1524 times (80.95% to 82.37% reduction, 81.77% on average). Both runs of HGA-SVM found the best fitness of 87.97% classification accuracy but the one using neighbor search was 5.79 times faster (61.42 mins v.s. 10.60 mins) per generation. The similar observation was also found with Adult dataset: 79.10% to 84.70% search reduction

by neighbor search, i.e., on average 5.76 times faster per generation.

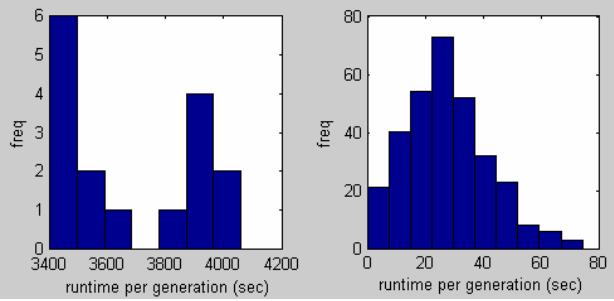

Figure 7. Integration of Four Improvement Strategies (Austrian) (left: standard GA-SVM, right: HGA-SVM)

Finally, the collective speedup of all four improvement strategies was evaluated. A remarkable reduction on computational cost was observed. Figure 7 shows the distributions of runtime per generation with Austrian dataset. The average runtime per generation is reduced from 61.42 min to 0.46 min, ~133 times.

In all the above experiments, the improvement was also observed to SVM learning accuracy over 10-fold cross validation (Figure 8). The learning accuracy was enhanced by 3.74% - 8.10% as a result of feature selection.

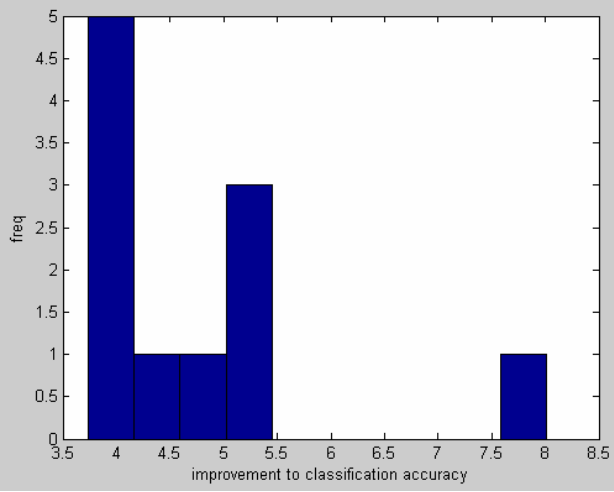

Figure 8. Improvement to Classification Accuracy

IV. Conclusions and Future Work

Our HGA-SVM combines parallelization and heuristic techniques that can effectively lower the computational cost in the feature selection model. We had demonstrated the individual speedup gain from parallel GA, parallel SVM, neighbor search and evaluation caching as well as their collective gain. Through the feature selection, the learning accuracy of SVM was enhanced as well. Overall, our HGA-SVM was proved to be useful in alleviating the computational cost with the improved learning performance, allowing the feasible application to larger data and more complex data

There is still space for the further development and research in HPC-enabled GA-SVM feature selection: 1) the present evaluation caching strategy was shown to be ineffective for high dimensional data. A possible solution could be applying distance-based fuzzy matching rather than exact matching to increase cache hit. 2) runtime may be further reduced by exploring more computational-efficient SVM algorithms, faster parameter tuning mechanism, and adaptive cross validation, etc. 3) the present efforts focus on runtime reduction with assumptions that the data should be containable in the memory. In order to deal with terabytes of data, the model might need some scalable tweaks such as boosting techniques. By committing both runtime reduction and storage stability, future research may aim at building a practical feature selection model based on HPC and heuristic techniques for solving the large-scale problems.

Reference

[1] B.E. Boser, I.M. Guyon, and V.N. Vapnik, "A training algorithm for optimal margin classifiers," *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory*, 1992, pp. 144--152.

[1] O. Chapelle, V. Vapnik, O. Bousquet, and S. Mukherjee, "Choosing Multiple Parameters for Support Vector Machines," *Machine Learning*, vol. 46, Jan. 2002, pp. 131-159.

[3] C.W. Hsu, C.C. Chang, and C.J. Lin, *A practical guide to support vector classification*, 2003.

[1] M. Mitchell, *An Introduction to Genetic Algorithms*, 1998.

[5] R. Tanese, "Distributed genetic algorithms," *Proceedings of the third international conference on Genetic algorithms*, George Mason University, United States: Morgan Kaufmann Publishers Inc., 1989, pp. 434-439.

[6] V.N. Vapnik, *Statistical Learning Theory*, Wiley-Interscience, 1998.

[7] I. Foster, *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*, Addison Wesley, 1995.

[8] J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," *Advances in Kernel Methods-Support Vector Learning*, 1999, pp. 185-208.

[9] C.C. Chang and C.J. Lin, "LIBSVM: a library for support vector machines," *Software available at http://www. csie. ntu. edu. tw/cjlin/libsvm*, vol. 80, 2001, pp. 604–611.

[10] L. Dagum and R. Menon, "OpenMP: An Industry-Standard API for Shared-Memory Programming," *IEEE COMPUTATIONAL SCIENCE & ENGINEERING*, 1998, pp. 46-55.

[11] M. Momma and K.P. Bennett, "A pattern search method for model selection of support vector regression," *IN PROCEEDINGS OF THE SIAM INTERNATIONAL CONFERENCE ON DATA MINING*, 2002.

[12] C.R. Houck, J. Joines, and M. Kay, "A Genetic Algorithm for Function Optimization: A Matlab Implementation," *NCSU-IE TR*, vol. 95, 1995.

[13] J.W. Eaton, "Octave," *http://www.gnu.org/software/octave/*.

[14] J. Fernández, M. Anguita, E. Ros, and J. Bernier, "SCE Toolboxes for the Development of High-Level Parallel Applications," *Computational Science – ICCS 2006*, 2006, pp. 518-525.