Journal of Computer Science 7 (5): 707-714, 2011
ISSN 1549-3636
© 2011 Science Publications

# Feature Subset Selection for Hot Method Prediction using Genetic Algorithm wrapped with Support Vector Machines

Sandra Johnson and Valli Shanmugam
Department of Computer Science and Engineering, College of Engineering,
Anna University, Guindy, Chennai, 600025, Tamil Nadu, India

**Abstract: Problem statement:** All compilers have simple profiling-based heuristics to identify and predict program hot methods and also to make optimization decisions. The major challenge in the profile-based optimization is addressing the problem of overhead. The aim of this work is to perform feature subset selection using Genetic Algorithms (GA) to improve and refine the machine learnt static hot method predictive technique and to compare the performance of the new models against the simple heuristics. **Approach:** The relevant features for training the predictive models are extracted from an initial set of randomly selected ninety static program features, with the help of the GA wrapped with the predictive model using the Support Vector Machine (SVM), a Machine Learning (ML) algorithm. **Results:** The GA-generated feature subsets containing thirty and twenty nine features respectively for the two predictive models when tested on MiBench predict Long Running Hot Methods (LRHM) and frequently called hot methods (FCHM) with the respective accuracies of 71% and 80% achieving an increase of 19% and 22%. Further, inlining of the predicted LRHM and FCHM improve the program performance by 3% and 5% as against 4% and 6% with Low Level Virtual Machines (LLVM) default heuristics. When intra-procedural optimizations (IPO) are performed on the predicted hot methods, this system offers a performance improvement of 5% and 4% as against 0% and 3% by LLVM default heuristics on LRHM and FCHM respectively. However, we observe an improvement of 36% in certain individual programs. **Conclusion:** Overall, the results indicate that the GA wrapped with SVM derived feature reduction improves the hot method prediction accuracy and that the technique of hot method prediction based optimization is potentially useful in selective optimization.

**Key words:** Feature selection, genetic algorithm, hot method prediction, support vector machine, virtual machine, Frequently Called Hot Methods (FCHM), Machine Learning (ML), Intra-Procedural Optimizations (IPO), Total Prediction Accuracy (TPA), Hot Method Prediction Accuracy (HMPA), Low Level Virtual Machine (LLVM)

## INTRODUCTION

Compiler optimizations are most effective when targeted at the hot methods of the input program. Method hotness, determined by execution time and call frequency, is still detected and predicted by profiling in both dynamic and static optimization systems. Although profiling is accurate, it incurs a lot of overhead which impedes program speed. The need for improving the accuracy of the hot method predictive models necessitates the focus on feature subset selection since feature selection greatly influences the performance of the machine learnt predictive models. The main aim of this work is to implement the machine learnt static hot method prediction technique using Genetic Algorithms (GA) derived feature subsets. By hot methods, we mean the long running and frequently called program segments that form the vital targets for various compiler optimization techniques (Sandra and Valli, 0000).

The relevant features are extracted from an initial set of randomly selected ninety static program features, using a GA (Koza, 1990) wrapped with the predictive model based on the Support Vector Machine (SVM) (Vapnik, 1997), a ML algorithm. The genetic algorithm proven to be an effective search tool is used in this work. The evaluation of features is based on the feedback obtained from the predictive models. Hence, the time required to converge at the final feature subset is dependent on the number of generations chosen in the GA.

The model's ability to achieve performance improvement is investigated by optimizing the predicted hot methods offline. The optimizations

**Corresponding Author:** Sandra Johnson, Department of Computer Science and Engineering, College of Engineering,
Anna University, Guindy Chennai, 600025, Tamil Nadu, India

applied are method inlining and Intra-Procedural Optimizations (IPO) like constant propagation and loop unrolling. The impact of optimizing the predicted hot methods on program performance is evaluated on UTDSP and MiBench benchmark programs. The results obtained are compared against LLVM's default optimization heuristics.

**Related work:** The application areas of the GA include a wide spectrum of problem solving domain such as supply chain management (Radhakrishnan *et al.*, 2009), input allocation problem (Madan *et al.*, 2010) and various feature selection problems. Several researchers ( Vafaie and Jong, 1992; Kohavi and John, 1997; Yang and Honavar, 1998; Pernkopf and O'Leary, 2001; Fröhlich and Chapelle, 2003; Yu and Cho, 2006; Huang and Wang, 2006; Faraoun and Rabhi, 2007; Rajavarman *et al.*, 2007; Ramirez and Puiggros, 2007; Xia *et al.*, 2009) have employed the genetic algorithm as a search tool in feature subset selection in their work. All these investigations have confirmed that the GA-generated feature subsets perform better than the initial universal set or the full feature set.

GA is particularly useful when search is large. Hence, the algorithm has found wide application in compiler research. To find the best optimization sequence which reduces the code size Cooper *et al*. (1999) have used the GA. Cavazos and O'Boyle (2005) have used the GA to tune dynamic compiler inlining heuristics. Li *et al*. (2008) and Zhuo *et al*. (2008) have used GA-based feature subset selection for optimizing the SVM parameters. In our present work on hot method prediction, the GA is used only as a feature selection algorithm and the prediction models use the default SVM parameters. Sandra *et al*. (Sandra and Valli, 2008a; 2008b) in their work have developed a basic hot method prediction model to predict the call frequency, whereas in this work two predictive models are built (Sandra and Valli, 0000; 2010) one based on call frequency and another the time spent in a method. In a previous work on hot method prediction (Sandra and Valli, 0000) the authors deal with the construction of an effective feature set from a full set of ninety randomly chosen static features using a 'knock-out' algorithm. Their model for the long running hot methods guided by twenty nine static features provides 68% prediction accuracy and the one for the frequently called hot methods yields 61% prediction accuracy on UTDSP and MiBench benchmark programs when trained with ten features.

The GA has been used in compiler based feature generation problems (Leather *et al*., 2009), where, each feature is a sentence in a grammar for the purpose of loop unrolling optimization. In the present study, we use the GA for selecting features specific to the prediction of hot methods and then apply inlining and intra-procedural optimizations to evaluate the effects of prediction. Stephenson *et al*. (2003), in their work, have used the GA to automatically search the solution space of the priority function, while we have used it for feature reduction.

## MATERIALS AND METHODS

In the construction of the predictive models, an initial set of ninety static program features (Sandra and Valli, 0000) has been used, for training and testing the classifiers. The SVM classifier is trained offline with the training dataset taken from the UTDSP and MiBench benchmark suites to predict hot methods of a new untrained program. Selecting the most relevant feature for a particular learning problem is a key challenge because any inappropriate feature included in the final set is bound to misguide the predictive models. Feature reduction from the full feature set is performed using the standard GA.

**Introduction to the genetic algorithm:** The genetic algorithm (Koza, 1990) is a powerful problem solving strategy that is widely used as a search tool for feature subset selection in any ML based classification problem. It works on the central evolutionary principle of the "survival of the fittest". Evolution is a population phenomenon and its forces operate on the individual's phenotypes that are manifestations of their genetic makeup called genotype. Based on their contribution to the individual's reproductive fitness they are either preserved or rejected during the selection process. The adaptive value of the phenotypes is influenced by the random variations introduced by the regular gene recombination effected by crossovers in chromosome segments and to a small extent the gene alterations introduced by point mutations. Individuals that are adaptively superior to others are selected to be the parents of the next generation. Over many generations of such progressive adaptation operating under selection pressure, a population that is far superior to the initial one appears. A GA is a programming technique that mimics this evolutionary mechanism to evaluate and select the best digital individual from among a pool of randomly generated candidates or solutions.

The crossover operator generates new offspring from parents by interchanging the genes between the parents at the crossover point. Crossovers can be single point, two point or homologue. Mutation alters the genes at random points to generate new offspring. Figure 1 shows these genetic operations.

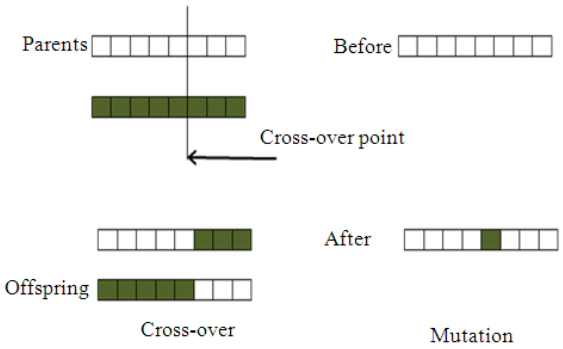


Fig. 1: Crossover and mutation operators in GA

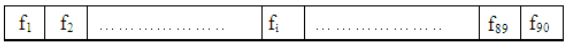


Fig. 2: Bitstring coding of chromosome

1. Initialize the genetic algorithm parameters as given in Table 1 and randomly generate the initial population of bitstrings.
2. Start the evolution process with the population.
   - 2.1 Using the bitstring representing the feature subset vector, for each program 'i' in the benchmark suite
     - 2.1.1 Generate the feature values for the training data set from the set of programs in the benchmark suite leaving the $i^{th}$ program and testing data set from the program 'i' (leave-one-out methodology).
     - 2.1.2 Train the SVM-based hot method prediction model.
     - 2.1.3 Test the hot method prediction for the $i^{th}$ program.
     - 2.1.4 Calculate the hot method prediction accuracy for the test program 'i'.
   - 2.2 Calculate the average hot method prediction accuracy for all the tested programs.
   - 2.3 Calculate the fitness function using the equation (1).
3. If the maximum fitness value is achieved or the number of generations is over, then stop the evolution process else go to step 2
4. The bitstring with the maximum fitness value is the resultant bit vector representing the feature subset.

Fig. 3: Algorithm for Feature Subset Selection using GA wrapped with SVM

**Genetic algorithm in feature subset selection:** An initial population of solutions is generated wherein each individual is represented by a chromosome. The feature vector consisting of a set of randomly selected ninety static program features is represented as a bitstring in the chromosome. Every bit is either '0' or '1' in the chromosome. The $f_i$ in Fig. 2 represents the mask value of the $i^{th}$ feature. A '1' includes the $i^{th}$ feature and a '0' excludes the respective feature.

The number of '1's in the bitstring represents the number of features selected. These bitstrings constitute the genotype which should be converted to its phenotype for evaluating its fitness value. The phenotype of the genes in the chromosome is the feature value extracted from each method in a benchmark program. For each individual in the initial population, the genotype of the chromosomes is translated into its phenotype. That is, the feature vectors for the training data set are created by extracting the individual feature values of each method from the set of training benchmark programs. The testing data set is also constructed using the same chromosome from each method in the test benchmark programs.

A predictive model is constructed based on the ML-based SVM algorithm, to predict hot methods. The prediction accuracy obtained is fed to the GA to evaluate the fitness of the chromosome. The GA uses a metric called a fitness function that evaluates the fitness of the bit string. The fitness criterion is designed on the basis of the hot method prediction accuracy and the number of features and is calculated using the formula given in Eq. 1:

$$\text{fitness} = (w_a * \text{prediction\_accuracy}) + (w_n * (\text{CHROMO\_LEN} - \sum_{i=1}^{\text{CHROMO\_LEN}} f_i)) \quad (1)$$

The $w_a$ in Eq. 1 represents the weight associated with the prediction accuracy and $w_n$ is the weight associated with the number of features. Those individuals which exhibit a higher fitness value than others are passed on to the next generation. In our scheme of searching strategy, a high fitness value is attributed to the chromosome when the prediction accuracy is high with a small number of features. The weights associated are changed for different runs and finally set to 50% both for $w_a$ and $w_n$. The CHROMO_LEN is the total number of ninety static features used in this work. The $f_i$ is the feature vector in bitstring form as given in Fig. 2.

Thus, the principle of the "survival of the fittest" of the GA is applied to retain the individuals with a high fitness value as "elitism" of the population to constitute the next generation. The fittest chromosomes representing the evolving individuals, survive the selection procedure. Two terminating conditions are used to stop the evolution process. One of them is the maximum fitness value and the other is the number of generations. If the terminating condition is reached, the evolution process stops and the individual with the highest fitness value is returned as the best solution. Else, the evolution process continues with the two genetic operators, namely, crossover and mutation. Two individuals with the highest fitness value are chosen for the crossover operation to produce two offsprings. Crossover points are chosen randomly. Mutation is also decided randomly and is applied on the two offsprings in the new generation. The process is repeated for the new generations of the population. The procedure for feature subset selection using GA wrapped with SVM is given in Fig. 3.

**Evaluation of hot method prediction accuracy:** To test and evaluate the performance accuracy of our prediction models, we use the standard 'leave-one-out' methodology under a subset of programs of MiBench (Guthaus *et al*., 2001) and UTDSP (Lee, 1998) benchmark suites that are successfully compiled in Low Level Virtual Machine (LLVM) (Lattner and Adve, 2004).

We use the following evaluation metrics (Sandra and Valli, 0000) to measure the performance of the predictive models. Total Prediction Accuracy (TPA) is the ratio of the number of correct predictions of both the hot and cold methods to the total number of methods in the program. The Hot Method Prediction Accuracy (HMPA) is defined as the ratio of the number of predicted hot methods to the total number of methods that are actually hot. Biased Hot Method Prediction Accuracy (BHMPA) is the ratio of the number of predicted hot methods to the total number of methods that are actually hot. Bias can be either 'hot' or 'cold' and the ML-based optimizing compiler optimizes all the methods in the program when the model is 'hot biased' and optimizes nothing if the model is cold biased. The bias factor is calculated using Eq. 2:

$$\text{Bias} = \frac{\text{BHMPA} - \text{TPA}}{100 - \text{HMT}} \tag{2}$$

To ascertain that the prediction values are accurate, we eliminate the bias using Eq. 3:

$$\text{HMPA} = \left(\text{BHMPA} - \left(|\text{Bias}| * \text{BHMPA}\right)\right) \tag{3}$$

A Hot Method Threshold (HMT) is set to find the actual number of methods that are hot in a program and a HMT of 50% is arbitrarily fixed for the evaluation of the predictive models. In the case of the Long Running Hot Methods (LRHM) predictive model, the 'gprof' tool is used to determine the execution time of each method, while profiling is used to find the call frequency of the methods for the Frequently Called Hot Methods (FCHM) predictive model during the training phase. The top 50% of the methods in both the models are designated as hot and assigned the label (+). The remaining methods are cold and are labeled (-1).

**Optimization effects:** Our goal is to show that the ML-based hot method prediction technique could be a viable alternative to the simple heuristics to decide when to apply inlining and Intra-Procedural Optimizations (IPO) for a new program. We compare the execution time of programs subjected to the selective optimization of hot methods predicted by the ML-based prediction models with LLVM's default optimization heuristics to assess the impact of the new approach on program performance.

## RESULTS

Genetic Algorithm Derived Feature Subsets: Based on the parameters given in Table 1, the GA is used to derive the bit string. The bit string is interpreted as a feature subset vector. If the bit position 'i' is '1' then the $i^{th}$ feature is chosen. All the features whose corresponding bit string positions are '0' are not included in the feature subset.

The fitness function uses the predictive model built by SVM to calculate the prediction accuracy of the feature subset. For each derived bit string, a predictive model is built and the Hot Method Prediction Accuracy (HMPA) is calculated on each UTDSP benchmark program using the standard 'leave-one-out' method. The fitness function is the average HMPA of all the UTDSP benchmark programs. The number of features, i.e., the number of '1's in the bit string is also used in the fitness function.

Figure 4 and 5 represent the derived feature vector for the LRHM and the FCHM predictive models with their respective thirty and twenty nine features. It is found that twenty one features are common to the two predictive models and only nine and eight features are unique to the LRHM and FCHM predictive models respectively. Table 2 gives the static feature subsets for the two predictive models generated by GA.

**Hot method prediction:** The GA derived feature subset of thirty and twenty-nine features are used in building the LRHM and FCHM predictive models. Table 3 presents the HMPA obtained from the UTDSP and the MiBench benchmark programs. The two benchmark suites are designed with programs that are successfully compiled in LLVM compiler infrastructure.

Table 1: Parameters used for the genetic algorithm

| Parameters | Values |
|---|---|
| Population size | 10.0 |
| Number of Generations | 50.0 |
| Mutation Rate probability | 0.1 |
| Chromosome length | 90.0 |

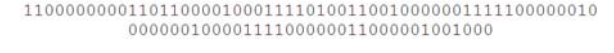


Fig. 4: Feature string for LRHM

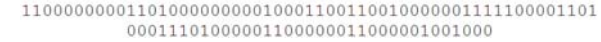


Fig. 5: Feature string for FCHM

Table 2: Subset of Features generated by the GA wrapped with SVM

| Features Common to LRHM and FCHM | | LRHM Features | |
|---|---|---|---|
| Feature No. | Static feature | Feature No. | Static feature |
| 1 | Number of Return Instruction | 15 | Number of Seteq Instruction |
| 2 | Number of Branch Instruction | 20 | Number of Setgt Instruction |
| 11 | Number of rem Instruction | 25 | Number of Store Instruction |
| 12 | Number of and Instruction | 26 | Number of GetElementPtr instruction |
| 14 | Number of Xor Instruction | 27 | Number of Phi node Instruction |
| 24 | Number of Load instruction | 54 | Number of constant global variables in a function |
| 29 | Number of CallInst in a method | 62 | Number of global variables which is a Null Value |
| 32 | Vanext instruction-used in llvm 1.5 and before | 67 | Number of derived type global variables |
| 33 | Vaarg instruction-used in llvm1.5 and before | 68 | Number of pointer type global variables |
| 36 | Number of Userop2 instruction | | |
| 43 | Number of instructions | | FCHM Features |
| 44 | Number of basic blocks | 28 | Number of Cast Instruction |
| 45 | Average number of incoming values in a phinode instruction | 52 | Number of back edges |
| 46 | Number of arguments | 53 | Number of defining edges |
| 47 | Is Variable argument? | 55 | Number of global variables having internal linkages |
| 69 | No of Array Global Variables | 59 | Number of global variables having link once linkages |
| 70 | No of Structure Global Variables | 60 | Number of global variables having weak linkages |
| 77 | Number of abstract global variables | 61 | Number of global variables having appending linkages |
| 78 | No of floating point array global variables | 63 | Number of global variables having initializers |
| 84 | Number of basic blocks with more than two predecessors | | |
| 87 | No of Basic Blocks with one successor And one predecessor | | |

Table 3: Prediction accuracy for LRHM and FCHM on UTDSP and MiBench benchmark programs

| | LRHM | | | | FCHM | | | |
|---|---|---|---|---|---|---|---|---|
| Benchmark | TPA (%) | BHMPA (%) | BIAS | HMPA (%) | TPA (%) | BHMPA (%) | BIAS | HMPA (%) |
| **UTDSP** | | | | | | | | |
| fft | 100 | 100 | 0.00 | 100 | 100 | 100 | 0.00 | 100 |
| fir | 100 | 100 | 0.00 | 100 | 62 | 75 | 0.26 | 56 |
| iir | 100 | 100 | 0.00 | 100 | 0 | 0 | 0.00 | 0 |
| latnrm | 100 | 100 | 0.00 | 100 | 100 | 100 | 0.00 | 100 |
| lmsfir | 100 | 100 | 0.00 | 100 | 100 | 100 | 0.00 | 100 |
| mult | 100 | 100 | 0.00 | 100 | 100 | 100 | 0.00 | 100 |
| adpcm | 100 | 100 | 0.00 | 100 | 93 | 100 | 0.14 | 86 |
| compress | 37 | 56 | 0.38 | 35 | 100 | 100 | 0.00 | 100 |
| edge_detect | 75 | 83 | 0.16 | 70 | 75 | 50 | -0.50 | 25 |
| G721.MarcusLee | 83 | 100 | 0.34 | 66 | 100 | 100 | 0.00 | 100 |
| G721.WendyFung | 91 | 100 | 0.18 | 82 | 86 | 81 | -0.10 | 73 |
| G722 | 75 | 85 | 0.20 | 68 | 75 | 50 | -0.50 | 25 |
| histogram | 83 | 100 | 0.34 | 66 | 100 | 100 | 0.00 | 100 |
| jpeg | 85 | 80 | -0.10 | 72 | 42 | 33 | -0.18 | 27 |
| lpc | 100 | 100 | 0.00 | 100 | 100 | 100 | 0.00 | 100 |
| spectral | 43 | 58 | 0.30 | 41 | 100 | 100 | 0.00 | 100 |
| trellis | 88 | 86 | -0.04 | 83 | 88 | 86 | -0.04 | 83 |
| V32.modem | 75 | 75 | 0.00 | 75 | 50 | 100 | 1.00 | 0 |
| UTDSP average | 85 | 90 | 0.10 | 81 | 82 | 82 | 0.00 | 71 |
| **MiBench** | | | | | | | | |
| basicmath | 66 | 66 | 0.00 | 66 | 100 | 100 | 0.00 | 100 |
| bitcount | 100 | 100 | 0.00 | 100 | 100 | 100 | 0.00 | 100 |
| qsort | 50 | 50 | 0.00 | 50 | 100 | 100 | 0.00 | 100 |
| susan | 94 | 100 | 0.12 | 88 | 84 | 75 | -0.18 | 62 |
| dijkstra | 100 | 100 | 0.00 | 100 | 100 | 100 | 0.00 | 100 |
| patricia | 100 | 100 | 0.00 | 100 | 100 | 100 | 0.00 | 100 |
| stringsearch | 100 | 100 | 0.00 | 100 | 100 | 100 | 0.00 | 100 |
| rijndael | 85 | 75 | -0.20 | 60 | 57 | 0 | -1.14 | 0 |
| sha | 100 | 100 | 0.00 | 100 | 100 | 100 | 0.00 | 100 |
| CRC32 | 0 | 0 | 0.00 | 0 | 100 | 100 | 0.00 | 100 |
| FFT | 50 | 25 | -0.50 | 13 | 83 | 50 | -0.66 | 17 |
| MiBench average | 77 | 74 | -0.05 | 71 | 93 | 84 | -0.18 | 80 |
| Overall average | 82 | 84 | 0.04 | 77 | 86 | 83 | -0.07 | 74 |

Table 4: Performance Improvement obtained on the predicted LRHM and FCHM of UTDSP and MiBench benchmark by optimization.

| | LRHM | | | | FCHM | | | |
|---|---|---|---|---|---|---|---|---|
| Benchmark | Inlined-LLVM default heuristics | IPO-LLVM default heuristics | Inlined-ML | IPO-ML | Inlined-LLVM default heuristics | IPO-LLVM default heuristics | Inlined-ML | IPO- ML |
| **UTDSP** | | | | | | | | |
| fft | 6 | -3 | 3 | 18 | 16 | 6 | 6 | 6 |
| fir | 29 | 23 | 20 | 17 | 4 | -8 | 8 | 8 |
| iir | 15 | -7 | 11 | 4 | 13 | 8 | 8 | 5 |
| latnrm | 10 | 0 | 13 | 10 | 3 | 10 | 6 | 13 |
| lmsfir | 12 | 0 | 9 | 9 | -6 | 6 | 6 | 3 |
| mult | 17 | 3 | 0 | 17 | 11 | 4 | 7 | 11 |
| adpcm | 3 | 3 | 3 | 3 | 4 | 3 | 0 | 3 |
| compress | -31 | 0 | 9 | 9 | 16 | 6 | 0 | -4 |
| edge_detect | 8 | 0 | 8 | 3 | 8 | 5 | 8 | 5 |
| G721.MarcusLee | 19 | 0 | 0 | 0 | 17 | 4 | 8 | 4 |
| G721.WendyFung | 23 | 4 | 4 | 12 | 4 | -13 | -9 | -9 |
| G722 | 0 | 0 | 0 | 0 | 4 | 4 | 0 | 0 |
| histogram | 6 | 3 | 3 | 6 | 10 | -7 | 7 | 3 |
| jpeg | 13 | 4 | 8 | 8 | 25 | 7 | 4 | 7 |
| lpc | -1 | -3 | -4 | -7 | 1 | -4 | -1 | -5 |
| spectral | 8 | 7 | 7 | 3 | 7 | 5 | 3 | 3 |
| trellis | -18 | -5 | -15 | -20 | 2 | 4 | -2 | -2 |
| V32.modem | -4 | -4 | 0 | 0 | 16 | 12 | 16 | 8 |
| UTDSP average | 6 | 1 | 4 | 5 | 9 | 3 | 4 | 3 |
| **MiBench** | | | | | | | | |
| basicmath | -2 | 2 | -2 | -4 | 0 | 2 | 4 | 2 |
| bitcount | 0 | 0 | 0 | 7 | 7 | 0 | 0 | 7 |
| qsort | 24 | 24 | 12 | 18 | 12 | 30 | 12 | 36 |
| susan | -64 | -66 | 2 | 1 | -66 | -1 | 2 | 2 |
| dijkstra | 1 | 0 | 0 | 0 | 1 | -1 | 0 | 0 |
| patricia | 7 | 2 | 0 | -2 | 7 | -5 | 5 | 2 |
| stringsearch | 0 | 0 | -20 | 0 | 13 | 0 | 7 | 13 |
| rijndael | -6 | 1 | 1 | 2 | -7 | 2 | 0 | 1 |
| sha | -3 | -4 | 1 | -1 | 3 | -4 | 1 | -33 |
| CRC32 | 25 | 8 | 17 | 8 | 34 | 0 | 17 | 17 |
| FFT | 9 | 0 | 2 | 11 | 8 | 0 | 9 | 2 |
| MiBench average | -1 | -3 | 1 | 4 | 1 | 2 | 5 | 5 |
| Overall average | 4 | 0 | 3 | 5 | 6 | 3 | 5 | 4 |

Our present study of hot method prediction involving the GA in feature reduction shows that the models can achieve 81-71% HMPA on the UTDSP benchmark suite for LRHM and FCHM. When the hot methods in the benchmark programs of MiBench are predicted using the feature subset derived from the UTDSP benchmark, the HMPA obtained by the respective predictive models for the LRHM and the FCHM are 71-80%.

**Results of optimization:** The effects of inlining and IPO using constant propagation on the predicted LRHM and FCHM are evaluated on the MiBench and UTDSP benchmark suites and the results are presented in Table 4. The IPO performed on the predicted LRHM and FCHM achieve an overall improvement of 5-4% respectively as against 0-3% using the LLVM's default set of optimization heuristics. However, inlining of LRHM and FCHM achieves 3-5% improvement on the program execution speed as against 4-6% seen in the case of LLVM's default heuristics. Despite a small decrease in the case of inlining, certain individual programs like 'latnrm' and 'susan' appear to have a positive impact. For instance, from Table 4, it is seen that 'latnrm' has a speedup of 13% when its LRHM are inlined and a speedup of 6% when its FCHM are inlined.

## DISCUSSION

In a previous work (Sandra and Valli, 0000), the authors have demonstrated that the predictive models for LRHM and FCHM trained with the full set of ninety features are capable of achieving 79-38% prediction accuracies on the UTDSP benchmark suites and 52-58% on MiBench. According to the present study, based on GA generated feature subsets of thirty and twenty-nine features to train the LRHM and FCHM models, the accuracies are 81-71% respectively on the

UTDSP and 71-80% on the MiBench. This is an improvement of 8-29% in predicting LRHM and FCHM over the models trained with the full feature set. In another approach (Sandra and Valli, 0000) where a 'knock-out' algorithm is implemented in order to eliminate irrelevant features, the LRHM and FCHM models have accuracies of 68- 61%. It is evident that GA derived predictive models provide improvement in prediction over the other models.

## CONCLUSION

This study describes the derivation of feature subsets using the GA wrapped with the ML-based algorithm SVM, to maximize the accuracy of the prediction of long running and frequently called hot methods, leading on to optimization and improvement in program performance. The GA evaluates the fitness of the feature subsets on UTDSP benchmark programs using the SVM algorithm. The GA-generated feature subsets containing thirty and twenty-nine features respectively for the two predictive models when tested on MiBench, predict the LRHM and FCHM with the respective prediction accuracies of 71-80%. The UTDSP benchmark suite achieves 81-71% for the LRHM and FCHM predictive models. These observations indicate that the GA-based approach in hot method prediction yields comparable results.

Future work in this GA based approach would focus on incorporating SVM parameters in the GA bit string coding.

## REFERENCES


Cavazos, J. and M.F.P. O'Boyle, 2005. Automatic tuning of inlining heuristics. Proceeding of the ACM/IEEE Conference on Supercomputing, Nov. 12-18, Seattle, WA, USA., pp: 14. DOI: 10.1109/SC.2005.14

Cooper, K.D., P.J. Schielke and D. Subramanian, 1999. Optimizing for reduced code space using genetic algorithms. Proceeding of the Workshop on Languages, Compilers and Tools for Embedded Systems, May 05-05, Atlanta, GA, USA., pp: 1-9. DOI: 10.1145/314403.314414

Faraoun, K.M. and A. Rabhi, 2007. Data dimensionality reduction based on genetic selection of feature subsets. J. Comput. Sci., 6: 9-19. http://www.dcc.ufla.br/infocomp/artigos/v6.2/art02.pdf

Fröhlich, H. and O. Chapelle, 2003. Feature selection for support vector machines by means of genetic algorithms. Proceeding of the 15th IEEE International Conference on Tools with Artificial Intelligence, Nov. 03-05, Sacramento, California, USA., pp: 142. DOI: 10.1109/TAI.2003.1250182

Guthaus, M.R., J.S. Ringenberg, D. Ernst, T.M. Austin, T. Mudge and R.B. Brown, 2001. MiBench: A free, commercially representative embedded benchmark suite. Proceedings of the IEEE 4th Annual Workshop on Workload Characterization, Dec. 02-02, Austin, TX, USA., pp: 3-14. ISBN: 0-7803-7315-4

Huang, C-L. and C-J. Wang, 2006. A GA-based feature selection and parameters optimization for support vector machines. Expert Syst. Appli., 31: 231-240. DOI: 10.1016/j.eswa.2005.09.024

Kohavi, R. and G.H. John, 1997. Wrappers for feature subset selection. Artificial Intell., 97: 273-324. DOI: 10.1016/S0004-3702(97)00043-X

Koza, J.R., 1990. Genetically breeding populations of computer programs to solve problems in artificial intelligence. Proceeding of the 2nd International Conference on Tools for AI, Nov. 6-9, IEEE Computer Society Press, Herndon, VA , USA., pp: 819-827. ISBN: 0-8186-2084-6

Lattner, C. and V. Adve, 2004. LLVM: A compilation framework for lifelong program analysis and transformation. Proceeding of the 2004 International Symposium on Code Generation and Optimization, Mar. 20-24, San Jose, CA, USA., pp: 75-86. DOI: 10.1109/CGO.2004.1281665

Leather, H., E. Bonilla and M. O'Boyle, 2009. Automatic feature generation for machine learning based optimizing compilation. Proceeding of the International Symposium on Code Generation and Optimization, Mar. 22-25, IEEE Computer Society, Seattle, Washington, pp: 81-91. DOI: 10.1109/CGO.2009.21

Lee, C., 1998. UTDSP benchmark suite. http://www.eecg.toronto.edu/~corinna/DSP/infrastructure/UTDSP.html

Madan, M. and S. Madan, 2010. Convalesce optimization for input allocation problem using hybrid genetic algorithm. J. Comput. Sci., 6: 413-416. DOI: 10.3844/jcssp.2010.413.416

Pernkopf, F. and P. O'Leary, 2001. Feature Selection for Classification using Genetic Algorithms with a Novel Encoding. CAIP 2001, LNCS 2124, Springer-Verlag, pp: 161-168. DOI: 10.1007/3-540-44692-3_20

Radhakrishnan, P., V.M. Prasad and M.R. Gopalan, 2009. Optimizing inventory using genetic algorithm for efficient supply chain management. J. Comput. Sci., 5: 233-241. DOI: 10.1.1.165.8114

Rajavarman, V.N. and S.P. Rajagopalan, 2007. Feature selection in data-mining for genetics using genetic algorithm. J. Comput. Sci., 3: 723-725. DOI: 10.1.1.165.9111

Ramirez, R. and M. Puiggros, 2007. A genetic programming approach to feature selection and classification of instantaneous cognitive states. Proceedings of the 2007 EvoWorkshops 2007 on EvoCoMnet, EvoFIN, EvoIASP,EvoINTERACTION, EvoMUSART, EvoSTOC and EvoTransLog: Applications of Evolutionary Computing, Springer-Verlag Berlin, Heidelberg, pp: 311-319. DOI: 10.1007/978-3-540-71805-5_34

Sandra, J. and S. Valli, 0000. Effective feature set construction for hot method prediction using support vector machines, under review-unpublished.

Sandra, J. and S. Valli, 2008a. An approach to predict hot methods using support vector machines. Proceedings of the Sixteenth International Conference on Advanced Computing and Communication (ADCOM'08), Dec. 14-17, Chennai, pp: 27-31. ISBN: 978-1-4244-2962-2

Sandra, J. and S. Valli, 2008b. Hot method prediction using support vector machines. Ubiquitous Comput. Commun. J., 3: 75-81. http://www.ubicc.org/files/pdf/HMPusingSVM_250.pdf

Sandra, J. and S. Valli, 2010. A relearning virtual machine for hot method prediction. Int. J. Soft Comput., 5: 206-210. DOI: 10.3923/ijscomp.2010.206.210

Stephenson, M., S. Amarasinghe, M. Martin, U. M. O'Reilly, 2003. Meta optimization: Improving compiler heuristics with machine learning. Proceeding of the ACM SIGPLAN Conference on Programming Language Design and Implementation, June 08-11, San Diego, CA, USA., pp: 77-90. DOI: 10.1145/781131.781141

Vafaie, H. and K.D. Jong, 1992. Genetic algorithms as a tool for feature selection in machine learning. Proceeding of the 1992 IEEE International Conference on Tools with AI, Nov. 10-13, Arlington, VA , USA., pp: 200-203. http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=246402

Vapnik, V.N., 1997. The Support Vector Method. Artificial Neural Networks. ICANN'97. LNCS 1327, Springer-Verlag, DOI: 10.1007/BFb0020166, 261-271

Xia, Z., Sun, X., Qin, J. and Niu, C., 2009. Feature selection for image steganalysis using hybrid genetic algorithm. inform. Technol. J., 8: 811-820. DOI: 10.3923/itj.2009.811.820

Yang, Y. and Honavar, 1998. Feature subset selection using a genetic algorithm. IEEE Intell. Syst., 13: 44-49. DOI: 10.1109/5254.671091

Yu, E. and S. Cho, 2006. Ensemble based on GA wrapper feature selection. Int. J. Comput. Indus. Eng., 51: 111-116. DOI: 10.1016/j.cie.2006.07.004

Zhuo, L., J. Zheng, F. Wang, X. Li, B. Ai and J. Qian, 2008. A genetic algorithm based wrapper feature selection method for classification of hyperspectral images using support vector machine. Int. Arch. Photogr., Remote Sens. Spatial Inform. Sci., 7147: 397- 402. DOI: 10.1117/12.813256