Hybrid Genetic Algorithms for Feature Selection

Il-Seok Oh, *Member*, *IEEE*, Jin-Seon Lee, and Byung-Ro Moon, *Member*, *IEEE*

Abstract—This paper proposes a novel hybrid genetic algorithm for feature selection. Local search operations are devised and embedded in hybrid GAs to fine-tune the search. The operations are parameterized in terms of their fine-tuning power, and their effectiveness and timing requirements are analyzed and compared. The hybridization technique produces two desirable effects: a significant improvement in the final performance and the acquisition of subset-size control. The hybrid GAs showed better convergence properties compared to the classical GAs. A method of performing rigorous timing analysis was developed, in order to compare the timing requirement of the conventional and the proposed algorithms. Experiments performed with various standard data sets revealed that the proposed hybrid GA is superior to both a simple GA and sequential search algorithms.

Index Terms—Feature selection, hybrid genetic algorithm, sequential search algorithm, local search operation, atomic operation, multistart algorithm.

1 Introduction

Feature selection is the problem of selecting a subset of d features from a set of D features based on some optimization criterion. The primary purpose of feature selection is to design a more compact classifier with as little performance degradation as possible. The features removed should be useless, redundant, or of the least possible use. It is well known that, for a problem of nontrivial size, the optimal solution is computationally intractable due to the resulting exponential search space and, hence, all of the available algorithms mostly lead to suboptimal solutions. Literatures on the subject of feature selection are abundant, presenting excellent tutorials [11], [28], proposing a taxonomy of feature selection algorithms [9], [3], and comparative studies [5], [9], [12].

Recently, interest in feature selection has been on the increase for several reasons. First, new applications dealing with vast amounts of data have been developed, such as data mining [24], [17], multimedia information retrieval [18], [16], [15], and medical data processing [26]. Since the fast processing of a large volume of data is critical in these applications for the purpose of real-time processing or to provide a quick response to users, limiting the number of features is a very important requirement. Feature selection is a prerequisite when using multiple sets of features, as this is required for the subsequent processing involving classification or clustering. Some examples include aerial photo interpretation [7], correspondence in stereo vision [8], and handwriting recognition [22].

- *I.-S. Oh is with the Division of Electronics and Computer Engineering, Chonbuk National University, Jeonju, Chonbuk 561-756, Korea. E-mail: isoh@chonbuk.ac.kr.*
- *J.-S. Lee is with the Department of Computer Engineering, Woosuk University, Samrye, Chonbuk 565-701, Korea. E-mail: jslee@woosuk.ac.kr.*
- *B.-R. Moon is with the School of Computer Science and Engineering, Seoul National University, Seoul 151-742, Korea. E-mail: moon@soar.snu.ac.kr.*

Manuscript received 9 July 2002; revised 12 Sept. 2003; accepted 8 Apr. 2004. Recommended for acceptance by W.T. Freeman.
For information on obtaining reprints of this article, please send e-mail to: tpami@computer.org, and reference IEEECS Log Number 116895.

Among the different categories of feature selection algorithms, the genetic algorithm (GA) is a rather recent development. The GA is biologically inspired and has many mechanisms mimicking natural evolution [6]. It has a great deal of potential in scientific and engineering optimization or search problems. Furthermore, GA is naturally applicable to feature selection since the problem has an exponential search space. The pioneering work by Siedlecki and Sklansky demonstrated evidence for the superiority of the GA compared to representative classical algorithms [29]. Subsequently, many literatures were published that have shown advantages of the GAs for feature selection [1], [30], [13], [27]. Although they presented experimental data supporting the superiority of GA, other literatures involving comparative studies did not give satisfactory arguments. Our review of those comparative studies and our own experimental results indicate that, to date, the SFFS (sequential floating forward search) algorithm is the best among the sequential search algorithms, but between the SFFS and GA no clear cut case can be made for which is the better of the two. A review and our comments on the previous comparative studies are given in Section 2.3.

The main reason for the above inconsistencies stems from the inherent nature of the GA itself, i.e., the fact that it has too many variations and parameters that need to be handled properly for reasonable or good performance to be obtained. Our review reveals that the specification of the GA implementation provided in the literature is insufficient for the readers to reproduce the program codes.

The limitations associated with a simple or pure GA (SGA) have been uncovered in many applications. Under normal conditions, solutions provided by a simple GA are likely to be inferior or comparable to classical heuristic algorithms. A practical and effective way of overcoming this limitation is to hybridize the GA, by incorporating domain-specific knowledge. Three ways of hybridizing are suggested in [4]: problem-specific encoding, the use of special genetic operators, and the incorporation of the good features of classical algorithms. Hybrid GAs (HGA) which follow these principles have been developed in diverse application areas and successful performance has been obtained [10], [2], [32].

This paper proposes a hybrid GA designed to solve the feature selection problem. The idea was originally introduced in [23]. The goal of this study is to develop a GA with improved capability that has leading-edge performance over the conventional algorithms. Our idea lies in the hybridizing of the GA by embedding local search operations into the simple GA. These local search operations move solutions towards local optima, and these improvements are accumulated over all of the generations, resulting in a significant improvement in the overall performance. A number of local search operations were devised for our hybrid GA. These operations are useful in describing our hybrid GAs as well as the conventional heuristic algorithms.

Through the experimental results obtained using various data sets, the performances of our HGA and other algorithms are compared, in terms of the accuracy of their classification and the computational time. We were able to conclude that our HGA is superior to the other algorithms in terms of accuracy, particularly for large-sized problems.

Section 2 defines several local search operations. It also reviews the prior comparative studies and provides our comments on them. In Section 3, we give a detailed specification of a simple GA designed for feature selection. Section 4 proposes a hybrid GA and describes a way of embedding the local search operations. A timing analysis is given in Section 5. Our experimental results and a discussion are presented in Section 6. Finally, Section 7 concludes the paper.

2 Feature Selection and Classical Algorithms

This section first defines the feature selection problem by describing the associated terminologies, notations, and basic local search operations. The algorithms are classified into three categories and described using the basic local search operations. At the end of this section, representative comparative studies are reviewed and our contributions are discussed.

2.1 Feature Selection Problem

The feature selection problem involves the selection of a subset of d features from a total of D features, based on a given optimization criterion. Let us denote the D features uniquely by distinct numbers from 1 to D, so that the total set of D features can be written as $\mathrm{U} = \{1, 2, \ldots, D\}$. X denotes the subset of selected features and Y denotes the set of remaining features. So, $\mathrm{U} = \mathrm{X} \cup \mathrm{Y}$ at any time. $\mathrm{J(X)}$ denotes a function evaluating the performance of X. J may evaluate either the accuracy of a specific classifier on a specific data set (e.g., the wrapper approach as in [14]) or a generic statistical measurement (e.g., the filter approach). The choice of evaluation function, J, depends on the particular application.

We introduce the basic operations that allow the search toward local optima during the feature selection process. In describing these operations, we do not show explicit parameters for the size of X and Y since they are implicitly clear. The operations rem^g and add^g are generalized operations that choose g features simultaneously. The size of the set, S, is denoted by $|\mathrm{S}|$.

Local search operations:

rem^g: Choose the least significant feature subset L of X such that $|\mathrm{L}| = g$ and $\mathrm{L} = \mathrm{argmax}_{\mathrm{A} \subset \mathrm{X}} \mathrm{J}(\mathrm{X} - \mathrm{A})$, and move all the features in L from X to Y.

add^g: Choose the most significant feature subset L of Y such that $|\mathrm{L}| = g$ and $\mathrm{L} = \mathrm{argmax}_{\mathrm{A} \subset \mathrm{Y}} \mathrm{J}(\mathrm{X} \cup \mathrm{A})$, and move all the features in L from Y to X.

$rem (= rem^1)$: Choose the least significant feature x in X such that $\mathrm{x} = \mathrm{argmax}_{\mathrm{a} \in \mathrm{X}} \mathrm{J}(\mathrm{X} - \{\mathrm{a}\})$, and move x to Y.

$add (= add^1)$: Choose the most significant feature y in Y such that $\mathrm{y} = \mathrm{argmax}_{\mathrm{a} \in \mathrm{Y}} \mathrm{J}(\mathrm{X} \cup \{\mathrm{a}\})$, and move y to X.

$REM(k)$: Repeat operation $rem\ k$ consecutive times.

$ADD(k)$: Repeat operation $add\ k$ consecutive times.

2.2 Feature Selection Algorithms

2.2.1 Enumeration Algorithms

- **Exhaustive Search**. All the possible $\binom{D}{d}$ subsets are evaluated and the best one among them is chosen. This guarantees the optimal solution, but the computational time is intractable when the problem size is not small.
- **Branch and Bound** [20], [31]. This algorithm generates a search tree that identifies the features being removed from the original set. It achieves a substantial reduction in the number of subset evaluations by pruning those subtrees that will never be superior to the current best solution. However, the main problem with this algorithm is its exponential time complexity. Additionally, this algorithm requires the strict assumption of monotonicity, i.e., adding new features never degrades the performance.

2.2.2 Sequential Search Algorithms

- SFS (sequential forward search) and SBF (sequential backward search).

```
Algorithm SFS:
  1. $\mathrm{X} = \Phi$; $\mathrm{Y} = \{i | 1 \le i \le D\}$;
  2. $ADD(d)$;

Algorithm SBS:
  1. $\mathrm{X} = \{i | 1 \le i \le D\}$; $\mathrm{Y} = \Phi$;
  2. $REM(D - d)$;

Algorithm Generalized-SFS:
  1. $\mathrm{X} = \Phi$; $Y = \{i | 1 \le i \le D\}$;
  2. repeat $add^g$ until $|\mathrm{X}| = d$;

Algorithm Generalized-SBS:
  1. $X = \{i | 1 \le i \le D\}$; $\mathrm{Y} = \Phi$;
  2. repeat $rem^g$ until $|\mathrm{X}| = d$;
```

- $\mathrm{PTA}(l, r)$ (plus-l and take away-r)

```
Algorithm PTA$(l, r)$ with $l > r$:
  1. $\mathrm{X} = \Phi$; $\mathrm{Y} = \{i | 1 \le i \le D\}$;
  2. repeat $\{ADD(l); REM(r);\}$ until $|\mathrm{X}| = d$;
```

```
Algorithm PTA($l,r$) with $l < r$:
  1. X = $\{i|1 \le i \le D\}$; Y = $\Phi$;
  2. repeat $\{REM(r); ADD(l);\}$ until $|X| = d$;

Algorithm Generalized-PTA($l,r$) with $l > r$:
  1. X = $\Phi$; Y = $\{i|1 \le i \le D\}$;
  2. repeat {
  3.    perform $add^g$ $l/g$ times;
  4.    perform $rem^g$ $r/g$ times;
  5. } until $|X| = d$;

Algorithm Generalized-PTA($l,r$) with $l < r$:
  1. X = $\{i|1 \le i \le D\}$; Y = $\Phi$;
  2. repeat{
  3.    perform $rem^g$ $r/g$ times;
  4.    perform $add^g$ $l/g$ times;
  5. } until $|X| = d$;
```

- SFFS (sequential forward floating search) and SBFS [25]. These algorithms cannot be compactly described using the local search operations. The operation *rem* captures the least significant feature with respect to the current subset, X, and immediately moves the feature from X to Y. However, SFFS and SFBS move the captured feature conditionally; so that these algorithms do not fix the values of l and r in PTA(l,r) in advance, but determine them dynamically. Also, they require the simultaneous existence of D temporary subsets denoted by X_k, $1 \le k \le D$, each containing k features. The X_k keeps track of the current best subset of size k. In line 2, more processing is allowed after the desired subset size is acquired, in order to obtain a better solution. When the condition is $k = D$, the best solutions are achieved.

```
Algorithm SFFS:
1. $X_0 = \Phi$; Y = $\{i|1 \le i \le D\}$; $k = 0$;  // initialization
2. while $(k < d + \delta)$ {
3.    y = $\mathrm{argmax}_{a \in Y - X_k} J(X_k \cup \{a\})$;
                              // find the most significant feature
4.    $X_{k+1} = X_k \cup \{y\}$; $k++$;
5.    x = $\mathrm{argmax}_{a \in X_k} J(X_k - \{a\})$;
                              // find the least significant feature
6.    while $(J(X_k - \{x\}) > J(X_{k-1}))$ {
7.       $X_{k-1} = X_k - \{x\}$; $k--$;
8.       x = $\mathrm{argmax}_{a \in X_k} J(X_k - \{a\})$;
                              // find the least significant feature
9.    }
10.}
```

2.2.3 Genetic Algorithms (GAs)

GA is a stochastic algorithm that mimics natural evolution. The most distinct aspect of this algorithm is that it maintains a set of solutions (called individuals or chromosomes) in a population. As in the case of biological evolution, it has a mechanism of selecting fitter chromosomes at each generation. To simulate the process of evolution, the selected chromosomes undergo genetic operations, such as crossover and mutation. The details are described in Section 3.

2.3 Comparative Studies and Discussions

Three typical comparative studies will be discussed. Ferri et al. asserted that SFFS is the best among the sequential search algorithms [5]. They also compared the SFFS and GA, and demonstrated that the performances of these two algorithms are comparable, but that the GA becomes inferior to the SFFS algorithm as the dimension (i.e., the value of D) increases. Jain and Zonker [9] concluded that SFFS is superior to other algorithms. Their experiments with the GA attained peak performance at the seventh or eighth generation, and they emphasized the difficulty of correctly setting the parameters when implementing GA. This is strong evidence of premature convergence. A recent work by Kudo and Sklansky [12] presented a rather fair comparison. They divided the problem into three categories, in terms of the problem size: small with $0 < D \le 19$, medium with $20 \le D \le 49$, and large with $50 \le D$. They concluded that SFFS is the best among the sequential search algorithms and is suitable for small and medium-sized problems, while the GA is suitable for large-sized problems. This argument contradicts that of Ferry et al. [5], and Kudo and Sklansky suspected that this contradiction is due to the different versions and implementation skills of GAs involved. Some papers claimed that GA is superior to other heuristic algorithms [29], [30], [27].

The above review leads us to make the following conclusions:

- SFFS is the best among the sequential search algorithms, but between the SFFS and GA, no clear cut case can be made for which is the better of the two. Although the superiority of GA was claimed in some literatures, more rigorous analyses are needed.
- Insufficient information was provided for the reproduction of the program code. Details and parameter settings should be provided to the readers so that they can obtain the same or similar performance.
- No serious attempts have been made to improve the capability of GA by incorporating domain knowledge.

The above factors provide the motivations for this study and the resolution of these problems is one of the most important contributions of this paper.

3 Simple GAs for Feature Selection

An important issue in designing a GA is the procedure used to control the genetic operations. Two alternatives are available and are shown below: the steady-state and generational procedures.

```
steady_state_GA()
{
   initialize population P;
   repeat {
     select two parents $p_1$ and $p_2$ from P;
     offspring = crossover($p_1, p_2$);
     mutation(offspring);
```

```
        replace(P, offspring);
    } until (stopping condition);
}

generational_GA()
{
    initialize population P;
    repeat {
      for ($i = 1$ to |P|) {
          select two parents $p_1$ and $p_2$ from P;
          $\mathrm{offspring}_i = \mathrm{crossover}(p_1, p_2)$;
          $\mathrm{mutation}(\mathrm{offspring}_i)$;
      }
      replace P with $\mathrm{offspring}_1, .., \mathrm{offspring}_{|P|}$;
    } until (stopping condition);
}
```

The most important difference between these two procedures lies in the method used to update the population. The steady-state procedure generally updates one chromosome in each generation, whereas the generational procedure updates the whole or most of the population. This difference has a significant impact on the behavior and power of the GA to find a solution in a large search space. The detailed specification of our GA is given below.

3.1 Chromosome Encoding

For the feature selection problem, a string with D binary digits is used. A binary digit represents a feature, values 1 and 0 meaning *selected* and *removed*, respectively. As an example, chromosome 00101000 means that the third and fifth features are selected. That is, the chromosome represents $X = \{3, 5\}$ and $Y = \{1, 2, 4, 6, 7, 8\}$.

3.2 Initial Population

The generation of the initial population is straightforward, as shown below. The function random_uniform() generates a random floating number within [0,1]. The expected number of selected features in an arbitrary initial solution is d.

```
Initial population:
    for ($i = 1$ to |P|)
        for (each gene $g$ in $i$th chromosome)
            if ($\mathrm{random\_uniform}() < d/D$) $g = 1$; else $g = 0$;
```

3.3 Fitness Evaluation, Selection, Replacement, and Stop

The evaluation is straightforward since a chromosome represents a selected feature subset, X, and the evaluation function is clear. In order to force a feature subset to satisfy the given subset size requirement, the size value, d, is taken as a *constraint* and a penalty is imposed on chromosomes breaking this constraint. The fitness of a chromosome C is defined as

$$\mathrm{fitness(C)} = \mathrm{J(X_C)} - \mathrm{penalty(X_C)},$$

where $\mathrm{X_C}$ is the corresponding feature subset of C, and $\mathrm{penalty(X_C)} = w^* ||\mathrm{X_C}| - d|$ with a penalty coefficient, w.

The chromosome selection for the next generation is done on the basis of fitness. The selection mechanism should ensure that fitter chromosomes have a higher probability of survival. Our design adopts the rank-based roulette-wheel selection scheme. The chromosomes in the population are sorted nonincreasingly in terms of their fitness, and the ith chromosome is assigned a probability of selection by a nonlinear function, $\mathrm{P}(i) = q(1-q)^{i-1}$. A larger value of q enforces a stronger selection pressure. The actual selection is done using the following roulette wheel procedure.

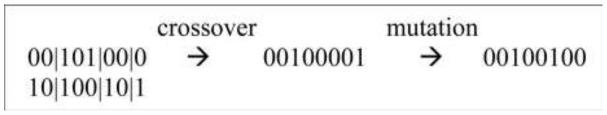

Fig. 1. An example of 3-point crossover and mutation.

Chromosome selection by roulette wheel:
1. Calculate accumulative probabilities for the ith chromosome by $p_i = \Sigma_{j=1,i}\mathrm{P}(j)$ for $i = 1, \ldots, n$ and $p_0 = 0$.
2. Generate a random number r within $[0, p_n]$.
3. Select the ith chromosome such that $p_{i-1} < r < p_i$.

* n: population size

For the steady state GA, we select two parent chromosomes using the above method. The crossover operation generates a new chromosome (offspring) out of the two parents, and the mutation operation slightly perturbs the offspring.

If the mutated chromosome is superior to both parents, it replaces the similar parent; if it is in between the two parents, it replaces the inferior parent; otherwise, the most inferior chromosome in the population is replaced. This replacement scheme was developed in [2]. The GA stops when the number of generations reaches the preset maximum generation T.

3.4 Crossover and Mutation

We use the standard crossover and mutation operators with a slight modification. An m-point crossover operator is used which chooses m cutting points at random and alternately copies each segment out of the two parents. The operations are exemplified in Fig. 1. The crossover may result in offspring that break the subset size requirement, because the exchanged gene segments may have different numbers of occurrences of 1.

The mutation is applied to the offspring. The mutation is likely to violate the subset size requirement, so we need to carefully control the numbers of 1-0 and 0-1 conversions. The following code forces the numbers to be similar, where the parameter p_m represents the mutation rate.

```
Controlled mutation:
  1. Let $n_0$ and $n_1$ to be numbers of 0-bits and 1-bits in the
     chromosome.
  2. $p_1 = p_\mathrm{m}$; $p_0 = p_\mathrm{m} \cdot n_1/n_0$;
  3. for (each gene $g$ in the chromosome)
  4.       Generate a random number $r$ within [0,1].
  5.       if($g = 1$ and $r < p_1$) convert $g$ to 0;
           else if($g = 0$ and $r < p_0$) convert $g$ to 1;
```

3.5 Parameters

No systematic parameter optimization process has so far been attempted, but the following parameter set was used in our experiments. Tuning the values to be suitable for a specific data set may give rise to improved performance.

Parameter setting:
Control procedure: steady-state
population size = 20
p_c (crossover probability) = 1.0 for steady-state (always applied), 0.6 for generational
p_m (mutation rate) = 0.1
q (in rank-based selection) = 0.25
w (penalty coefficient) = 0.5
T (maximum generation): dependent on data set

4 Hybrid GA

Genetic algorithms are able to escape from local optima by means of the crossover and mutation operators, and to explore a wide range of search space when the selection pressure is properly controlled. However, they are weak in fine-tuning near local optimum points, and this results in their having a long running time. To improve the fine-tuning capability of simple GAs, hybrid GAs have been developed in many applications, including the traveling salesman problem [10], graph partitioning problem [2], and image compression [32]. In a hybrid GA, chromosomes are improved by proper local search operations. We propose a hybrid GA for the feature selection problem.

The basic idea behind our hybrid GA (HGA) is to embed the problem-specific local search operations in a GA. The steady-state procedure incorporating this idea is outlined below.

```
HGA()
{
  initialize population P;
  **repeat**{
    select two parents $p_1$ and $p_2$ from P;
    offspring = crossover($p_1$, $p_2$);
    mutation(offspring);

    *local-improvement(offspring)*;

    replace(P, offspring);
  } **until** (stopping condition);
}

local_improvement(C) /* C: a chromosome */
{
  put features of 1-bits in C into X;
  put features of 0-bits in C into Y;

  **switch**{
    **case** $|X| = d$: $ripple\_rem(r)$; $ripple\_add(r)$;
    **case** $|X| < d$: repeat $ripple\_add(r)$ $d - |X|$ times;
    **case** $|X| > d$: repeat $ripple\_rem(r)$ $|X| - d$ times;
  }

  set bits in C for features in X to be 1;
  set bits in C for features in Y to be 0;
}
```

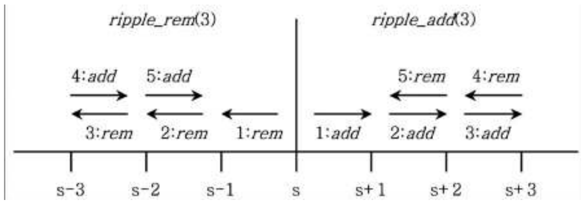

Fig. 2. Sequence of *add* s and *rem* s generated by the $ripple\_rem(3)$ and $ripple\_add(3)$.

It is probable that the offspring generated after the crossover and mutation operations will inherit the parents' good characteristics and be either superior or inferior to their parents. In our hybrid GA, before the replacement stage, the offspring is given the chance to be improved by means of local search operations.

A major concern in realizing the above idea is related to the problem of choosing the proper operations for local improvement. It is natural that we use the operations defined in Section 2.1: rem, add, $REM(k)$, and $ADD(k)$. The operations work with a current feature subset, X, represented by a chromosome, and they perform local searches around the current solution by removing the least significant features or adding the most significant features. By applying them in a proper sequence, we can accomplish local improvement of the chromosome. In addition, we can control the subset size by designing a proper sequence of operations. In other words, removing or adding features can be regarded as *repairing* operators.

The local improvement of a chromosome is accomplished using the code shown above. First, it converts the chromosome, C, into two feature subsets, X and Y. To describe the algorithm briefly, we define two local search operations. The $ripple\_rem(r)$ procedure first removes the least significant features r times and then adds the most significant features $r-1$ times, resulting in the removal of one feature. Likewise, $ripple\_add(r)$ increases the size of X by 1. We call the parameter r the *ripple factor*. Fig. 2 shows two examples of local search operations.

Local search operations:
$ripple\_rem(r) \equiv \{REM(r); ADD(r-1);\}$, $r \geq 1$
$ripple\_add(r) \equiv \{ADD(r); REM(r-1);\}$, $r \geq 1$

Depending on the number of features in X and subset size requirement, the algorithm handles three different cases, as illustrated in the code.

1. The size requirement is satisfied: X is perturbed by applying $ripple\_rem(r)$ and $ripple\_add(r)$.
2. There are fewer features in X than required: X is increased by applying $ripple\_add(r)$ a number of times.
3. There are more features in X than required: X is decreased by applying $ripple\_rem(r)$ a number of times.

The ripple factor is used to control the strength of local improvement. Although the reduction in the number of features caused by $ripple\_rem(r)$ is 1, independent of r, the ripple factor directly influences the actual number of rem and add operations to be executed. The larger the ripple factor, r, the stronger the local improvement we obtain. For example, $ripple\_rem(2)$ is accomplished by two rems and one add and it is evidently stronger than $ripple\_rem(1)$ which performs only one rem.

We obtain two beneficial effects that are useful in solving the feature selection problem. The first effect concerns the overall performance improvement obtained from the local improvement of chromosomes. Because we use the basic operations of rem and add that seek the least significant and the most significant features, a chromosome tends to improve its performance locally. The second beneficial effect concerns the control of the subset size. By conducting a sequence of rem and add operations, we can easily repair a subset, in order for it to satisfy the subset-size requirement.

Local search operations require considerable processing time. A larger ripple factor requires more computation time. On the other hand, the GA converges faster due to the fine-tuning power of local searches. One of the crucial issues in designing an efficient hybrid GA is how to speed up the local search operations, while minimizing the number of operations to be executed.

5 Timing Analysis and Efficiencies

Rigorous timing analysis helps us to perform a fair comparison of various algorithms and to estimate the actual computation times. Most conventional literatures are somewhat weak with regard to this matter. As an example, Kudo and Sklansky used the number of subset evaluations and big-O notation [12]. This kind of analysis is not very helpful, because the evaluations of subsets with different sizes consume significantly different amounts of computation time.

5.1 Analysis for Local Search Operations

Our analysis is rigorous in the sense that it uses *atomic* operations requiring a fixed amount of CPU cycles. Every feature selection algorithm consists of two major tasks: a control process managing X and Y and a subset evaluation process. The control process is computationally trivial and negligible, and most of the time is consumed by the subset evaluation process. Let $\mathrm{t}(s)$ be the computation time required to evaluate a feature subset, X, with size s. The value of $\mathrm{t}(s)$ depends not only on s, but also on the size of the training sample set, when we use a wrapper approach to subset evaluation. Since the size of the sample sets is fixed in advance for a given feature selection job, it can be regarded as constant. Therefore, $\mathrm{t}(s)$ depends only on the value of s. The evaluation of a single feature is called an *atomic operation*, and $\mathrm{t}(1)$ for the atomic operation is referred to as the *atomic time*.

For the simplicity of analysis, a linearity assumption is made, i.e., $\mathrm{t}(s) \cong s\sigma$ where σ represents the atomic time. The linearity assumption holds for k-NN classifiers, since the dominant operation is the distance calculation that is linear to the number of features. This assumption also holds for a neural network classifier, MLP (multiple layer perceptron) since both the forward classification and backward learning processes have $\mathrm{O}(s)$ time complexity.

Now, we analyze the computation time for the local search operations. $\mathrm{T}(op)$ denotes the timing requirement for the operation, op, measured in terms of the number of atomic operations. In this notation, the current size of X is s. Fig. 3 visually illustrates an example set of timing data for a

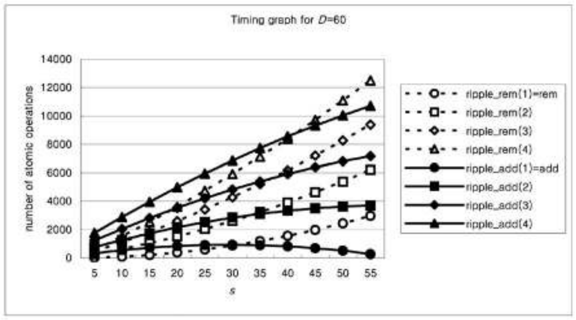

Fig. 3. Timing graphs for a problem with $D = 60$.

problem with $D = 60$. Note that the operations rem and add are the same as $ripple\_rem(1)$ and $ripple\_add(1)$, respectively. $\mathrm{T}(rem)$ has its peak at $s = D$, while $\mathrm{T}(add)$ has its peak at $s = D/2$. For ripple factors $r \geq 2$, the peaks are located at $s = D$. As the ripple factor increases, the gap between $\mathrm{T}(ripple\_rem(r))$ and $\mathrm{T}(ripple\_add(r))$ decreases. We note that $ripple\_rem(r)$ and $ripple\_add(r)$ consume approximately r times the number of atomic operations consumed by rem and add, respectively.

Timing analysis for local search operations:

$$\mathrm{T}(rem) = \mathrm{t}(s-1) \cdot s = \sigma(s^2 - s)$$
$$\mathrm{T}(add) = \mathrm{t}(s+1) \cdot (D-s) = \sigma(D \cdot s - s^2 - s + D)$$
$$\mathrm{T}(ripple\_rem(r)) = \Sigma_{s'=s-r+1,s}\mathrm{t}(s'-1) \cdot s' + \Sigma_{s'=s-r,s-2}\mathrm{t}(s'+1) \cdot (D-s')$$
$$\mathrm{T}(ripple\_add(r)) = \Sigma_{s'=s,s+r-1}\mathrm{t}(s'+1) \cdot (D-s') + \Sigma_{s'=s+2,s+r}\mathrm{t}(s'-1) \cdot s'$$
$$\mathrm{T}(ripple\_rem(1)) = \sigma(s^2 - s)$$
$$\mathrm{T}(ripple\_rem(2)) = \sigma(D \cdot s + s^2 - s - D)$$
$$\mathrm{T}(ripple\_rem(3)) = \sigma(2D \cdot s + s^2 - s - 3D)$$
$$\mathrm{T}(ripple\_rem(4)) = \sigma(3D \cdot s + s^2 - s - 6D)$$
$$\mathrm{T}(ripple\_add(1)) = \sigma(D \cdot s - s^2 - s + D)$$
$$\mathrm{T}(ripple\_add(2)) = \sigma(2D \cdot s - s^2 - s + 3D)$$
$$\mathrm{T}(ripple\_add(3)) = \sigma(3D \cdot s - s^2 - s + 6D)$$
$$\mathrm{T}(ripple\_add(4)) = \sigma(4D \cdot s - s^2 - s + 10D)$$

We can classify the feature selection algorithms into two categories: deterministic (enumeration algorithms and sequential search algorithms) and stochastic (GA). A deterministic algorithm performs the same sequence of operations and always produces the same result. On the other hand, a stochastic algorithm executes a different sequence of operations from one run to another, and produces a different solution on each occasion. If more CPU time is allowed, stochastic algorithms are likely to produce better solutions. A simple way of giving more time to a GA is to increase the generation number or the population size. This aspect of the GA is advantageous in most circumstances. Since feature selection is usually done in offline mode, CPU time is not tightly restricted. The GA is also inherently parallel, since the chromosomes in a population can be evaluated in parallel.

5.2 Time Saving by Bookkeeping

As mentioned above, the dominant operation of any feature selection algorithm is the evaluation of the feature subsets. Since these algorithms repeatedly add and remove features, it is possible that the same subsets are re-evaluated. By bookkeeping the already evaluated subsets and their

TABLE 1
Data Sets Used for Our Experiment

	Number of samples	Number of features	Number of classes	Evaluation method	
				Classifier	*x* in leave-*x*-out
Glass*	214	10	7	1-NN	1
Vowel*	990	10	11	1-NN	1
Wine*	178	13	3	1-NN	1
Letter*	20000	16	26	1-NN	5000
Vehicle*	846	18	4	1-NN	1
Segmentation*	210/2100***	19	7	1-NN	-
WDBC*	569	30	2	1-NN	1
Ionosphere*	351	34	2	1-NN	1
Satellite*	4435/2000***	36	6	1-NN	-
Sonar*	208	60	2	1-NN	1
Numerals**	4000/2000***	100	10	1-NN	-

* Downloaded from the UCI repository [19].
** The *gray-mesh* features were extracted from CENPARMI handwritten numeral samples [21].
*** *x/y* indicates that *x* training and *y* test samples are available. The whole training samples were used as prototypes for 1-NN.

performance data, we can avoid duplicate computations and, thus, speed up the algorithms.

To assess the effectiveness of the bookkeeping operation, we can use the hit ratio, which is defined as the probability that the subsets currently being evaluated are found in the book. However, this is not precise, for the same reason as that mentioned at the beginning of Section 5. We use the atomic operations in formulating the time saving ratio, e, and the speedup factor, p. The atomic operations related to the actually evaluated subset are referred to as *pure* and those that are skipped when the same subsets recur are called *duplicate*.

$$e = N_{dup}/(N_{pure} + N_{dup}) \text{ and } p = 1/(1-e),$$

where N_{pure} and N_{dup} represent the number of pure and duplicate atomic operations, respectively.

The SFS algorithm always proceeds in the forward direction, so duplicate evaluation never occurs. The PTA and SFFS algorithms proceed in both the forward and backward directions repeatedly, so duplicate evaluations are likely to occur. Since the GAs generate new chromosomes through the genetic operators, duplicate evaluation can occur. The hybrid GAs have the operations $ripple\_rem$ and $ripple\_add$ that repeat the rems and adds sequentially, so they are more likely to perform duplicate evaluations than a simple GA.

6 Experimental Results and Discussions

6.1 Environment

The 11 data sets used to test the algorithms are summarized in Table 1. Ten data sets were chosen from the UCI repository [19] with the condition that the number of features be 10 or greater, that there be no missing feature, and that all of the features be numeric. The other data set was obtained by extracting the *gray-mesh* features from the CENPARMI handwritten numeral samples [21]. To extract the gray-mesh features, a sample numeral image was size-normalized to 10*10 mesh and each of the pixel values in the mesh was taken as a feature value. All of the 11 data sets have been widely used in the pattern recognition community. The sources of the data sets are diverse, including character recognition, speech recognition, object recognition, and medical diagnosis. Additionally the dimensionalities covered by the data sets have a large spectrum ranging from 10 to 100.

Table 1 shows the specification of the data sets. For the sake of the performance analysis, we follow the categorization of problem sizes described by Kudo and Sklansky [12]: small with $0 < D \leq 19$, medium with $20 \leq D \leq 49$, and large with $50 \leq D$. The small-sized data sets include Glass, Vowel, Wine, Letter, Vehicle, and Segmentation. The next three data sets, WDBC, Ionosphere, and Satellite, are medium-sized. The last two data sets, Sonar and Numeral, are large-sized.

The evaluation was conducted in terms of two criteria: classification accuracy and computation time. Accuracy was measured by determining the recognition rate of 1-NN classifiers without rejection, and time was measured by determining the number of atomic operations. Eight algorithms were tested: SFS, $\mathrm{PTA}(l = 2, r = 1)$, SFFS, simple GA (SGA), and four hybrid GAs (HGAs) with different ripple factors. The HGAs are referred to as $\mathrm{HGA}(r)$, where r represents the ripple factor.

The steady-state version of the GA procedure was used for the SGA and HGA. The number of generations for the SGA and HGAs (i.e., T in the stopping condition) was controlled, such that the four HGAs use similar amounts of atomic operations and the SGA uses more than the HGAs. Table 2 shows the actual values for T. All of the data sets used the same values of T for the four HGAs. The recognition rates and timing data were collected with various values of d. The SFFS has an option related to the parameter δ in line 2 of the algorithm SFFS described in Section 2.2.2. We set the condition to be $k = D$, so that the best possible results are obtained by SFFS. For GA, five independent runs were executed and their average and maximum performances were presented.

6.2 Classification Accuracy

Table 3 summarizes the performance of the nine feature selection algorithms for the 11 data sets. For each of the data

TABLE 2
Actual Values for T in the Stopping Condition

	SGA	HGA(1)	HGA(2)	HGA(3)	HGA(4)
Glass	10000	600	200	120	80
Vowel	10000	600	200	120	80
Wine	10000	600	200	120	80
Letter	10000	600	200	120	80
Vehicle	10000	600	200	120	80
Segmentation	10000	600	200	120	80
WDBC	20000	600	200	120	80
Ionosphere	25000	600	200	120	80
Satellite	25000	600	200	120	80
Sonar	40000	600	200	120	80
Numerals	100000	600	200	120	80

sets, the recognition rates were measured for four values of d: $D/5$, $2D/5$, $3D/5$, and $4D/5$. For the data sets with low-dimensional feature spaces, the optimal solutions were provided as a result of an exhaustive search. In the case of the GAs, the average (x) and maximum (y) rates of the five runs are denoted by $x(y)$. The bold-faced figures represent the best results.

First, the effectiveness of the hybridization is analyzed. For this purpose, the five GAs are compared. In the case of the small-sized data sets, all of the GAs found the optimal solutions with only one exception: SGA with $d = 11$ in Vehicle. Since the average and maximum rates are the same for most cases, these small-sized data sets seem to be easy for the GAs. For the few cases that the SGA could not find the optima, the HGA found them.

For the medium-sized data sets (WDBC, Ionosphere, and Satellite), the best solutions were always found by the HGAs and some of them were also found by the SGA.

In the case of the large-sized data sets (Sonar and Numeral), the SGA found no best solutions and the HGAs outperformed the SGA. It is also notable that $\mathrm{HGA}(r \geq 2)$ were superior to HGA(1). In the case of the Sonar data set, the solutions obtained by $\mathrm{HGA}(r \geq 2)$ were better than the ones obtained by HGA(1) by about 0.5-1.0 percent. Additionally, with respect to the gaps between the average and maximum rates, $\mathrm{HGA}(r \geq 2)$ gave rise to less deviations than HGA(1). That is, $\mathrm{HGA}(r \geq 2)$ was more attractive than HGA(1) in terms of both solution quality and stability. More rigorous comparisons among the GAs will be made in Section 6.4, using their convergence characteristics.

Among the three sequential search algorithms, SFFS produced the best overall solutions with only a few exceptions. The PTA always outperformed SFS, and SFFS always outperformed PTA with only two exceptions, i.e., when $d = 27$ in Ionosphere and $d = 48$ in Sonar. For the small-sized data sets of Glass and Segmentation, SFS also produced the optimal solutions. It is worth noting that the gaps among the three algorithms grew with increasing problem size. As an example, using Sonar, SFFS produced better solutions than PTA by about 1.5-3 percent and PTA was better than SFS by about 0.5-3.5 percent.

Next, we compare the GAs with the sequential search algorithms. Since SFFS outperformed PTA and SFS, only SFFS was used for comparison. Overall, the SGA produced solutions comparable to those of SFFS on average. When considering the best solutions, SGA seemed more attractive than SFFS. The HGAs considerably improved the results of SGA. The HGAs were thus superior to SFFS in every case. In addition, the gaps between the GAs and SFFS become larger as the problem size increases.

6.3 Computation Time

Four data sets were chosen to analyze the computational times of the algorithms. The timing data used for them are illustrated in Table 4. The data used for the GAs are the averages of five runs. Since we used the condition $k = D$ for stopping SFFS, the data used with SFFS are the same for all values of d. Two data were measured in the form $N(e)$, where N represents the total number of atomic operations (i.e., $N_{pure} + N_{dup}$) and e denotes the time-saving ratio defined in Section 5.2. As an example, HGA(2) produced the data $N = 0.42\mathrm{M}$ and $e = 0.28$ to select a subset with $d = 12$ in Sonar. So, HGA(2) performed about 302K (= 0.42M*0.72) pure atomic operations and skipped about 117K (= 0.42M*0.28) duplicate atomic operations. The speedup factor, p, is 1.39 in this case.

Since SFS always proceeds in the forward direction, no redundant subset evaluation occurs and, thus, e is always 0. The PTA and SFFS obtained significant speedups by a factor of about $1.1 \sim 1.7$. The GAs showed large variations in the value of e depending on the size of the search spaces defined by D and d. As D increased, e decreased substantially. The factor d influenced e in the same manner. The large search spaces in the cases where $d = 24$ and 36 in Sonar and $d = 40$ and 60 in Numeral resulted in negligible speedups in the case of SGA and HGA(1). However, $\mathrm{HGA}(r \geq 2)$ showed notable speedups for the same cases. We noted a trend that large ripple factors led to greater speedups.

The SFFS consumed the longest time among the sequential search algorithms. The GAs consumed more time than the sequential algorithms due to their stochastic nature. According to Table 4, the GAs consumed about 3-6 times the computation time of SFFS for the large-sized data sets. Note that in SGA the time increased linearly with d since the same amount of computation is required for every generation. However, in the HGAs the time did not

TABLE 3
Recognition Rates for the 11 Data Sets (Unit: %)

Datasets	d*	Optimum	SFS	PTA	SFFS	SGA	HGA(1)	HGA(2)	HGA(3)	HGA(4)
Glass (*D*=10)	2	**99.07**	**99.07**	**99.07**	**99.07**	**99.07**(99.07)	**99.07**(99.07)	**99.07**(99.07)	NA	NA
	4	**100**	**100**	**100**	**100**	**100**(100)	**100**(100)	**100**(100)	**100**(100)	**100**(100)
	6	**100**	**100**	**100**	**100**	**100**(100)	**100**(100)	**100**(100)	**100**(100)	**100**(100)
	8	**100**	**100**	**100**	**100**	**100**(100)	**100**(100)	**100**(100)	**100**(100)	NA
Vowel (*D*=10)	2	**62.02**	**62.02**	**62.02**	**62.02**	**62.02**(62.02)	**62.02**(62.02)	**62.02**(62.02)	NA	NA
	4	**92.83**	92.63	**92.83**	**92.83**	**92.83**(92.83)	**92.83**(92.83)	**92.83**(92.83)	**92.83**(92.83)	**92.83**(92.83)
	6	**98.79**	98.28	**98.79**	**98.79**	**98.79**(98.79)	**98.79**(98.79)	**98.79**(98.79)	**98.79**(98.79)	**98.79**(98.79)
	8	**99.70**	**99.70**	**99.70**	**99.70**	**99.70**(99.70)	**99.70**(99.70)	**99.70**(99.70)	**99.70**(99.70)	NA
Wine (*D*=13)	3	**93.82**	**93.82**	**93.82**	**93.82**	**93.82**(93.82)	**93.82**(93.82)	**93.82**(93.82)	**93.82**(93.82)	NA
	5	**95.51**	94.38	94.38	94.94	**95.51**(95.51)	**95.51**(95.51)	**95.51**(95.51)	**95.51**(95.51)	**95.51**(95.51)
	8	**95.51**	**95.51**	**95.51**	**95.51**	**95.51**(95.51)	**95.51**(95.51)	**95.51**(95.51)	**95.51**(95.51)	**95.51**(95.51)
	10	**92.70**	92.13	92.13	**92.70**	**92.70**(92.70)	**92.70**(92.70)	**92.70**(92.70)	**92.70**(92.70)	**92.70**(92.70)
Letter (*D*=16)	3	**47.09**	**47.09**	**47.09**	**47.09**	**47.09**(47.09)	**47.09**(47.09)	**47.09**(47.09)	**47.09**(47.09)	NA
	6	**87.60**	86.20	**87.60**	**87.60**	**87.60**(87.60)	**87.60**(87.60)	**87.60**(87.60)	**87.60**(87.60)	**87.60**(87.60)
	10	**96.35**	96.12	**96.35**	**96.35**	**96.35**(96.35)	**96.35**(96.35)	**96.35**(96.35)	**96.35**(96.35)	**96.35**(96.35)
	13	**96.42**	**96.42**	**96.42**	**96.42**	**96.42**(96.42)	**96.42**(96.42)	**96.42**(96.42)	**96.42**(96.42)	**96.42**(96.42)
Vehicle (*D*=18)	4	**69.74**	62.77	64.78	69.15	69.50(69.74)	**69.74**(69.74)	**69.74**(69.74)	69.62(69.74)	69.39(69.74)
	7	**73.52**	69.15	70.09	**73.52**	72.97(73.52)	**73.52**(73.52)	**73.52**(73.52)	**73.52**(73.52)	**73.52**(73.52)
	11	**72.46**	69.50	71.75	71.87	71.84(71.87)	**72.46**(72.46)	**72.46**(72.46)	**72.46**(72.46)	72.29(72.46)
	14	**70.80**	68.20	**70.80**	**70.80**	**70.80**(70.80)	**70.80**(70.80)	**70.80**(70.80)	**70.80**(70.80)	**70.80**(70.80)
Segmen-tation (*D*=19)	4	**92.81**	**92.81**	**92.81**	**92.81**	**92.81**(92.81)	**92.81**(92.81)	**92.81**(92.81)	**92.81**(92.81)	**92.81**(92.81)
	8	**92.95**	**92.95**	**92.95**	**92.95**	**92.95**(92.95)	**92.95**(92.95)	**92.95**(92.95)	**92.95**(92.95)	**92.95**(92.95)
	11	**92.95**	**92.95**	**92.95**	**92.95**	**92.95**(92.95)	**92.95**(92.95)	**92.95**(92.95)	**92.95**(92.95)	**92.95**(92.95)
	15	**92.57**	**92.57**	**92.57**	**92.57**	**92.57**(92.57)	**92.57**(92.57)	**92.57**(92.57)	**92.57**(92.57)	**92.57**(92.57)
WDBC (*D*=30)	6	**94.90**	93.15	93.15	94.20	93.67(93.67)	93.92(94.90)	**94.38**(94.90)	93.99(94.20)	93.99(94.20)
	12	NA	92.62	92.97	94.20	93.95(94.38)	94.06(94.38)	94.06(94.38)	94.06(94.38)	**94.27**(94.38)
	18	NA	94.02	**94.20**	**94.20**	93.85(93.85)	93.92(94.20)	93.99(94.20)	94.13(94.20)	93.99(94.20)
	24	NA	92.44	93.50	**93.85**	**93.85**(93.85)	**93.85**(93.85)	**93.85**(93.85)	**93.85**(93.85)	**93.85**(93.85)
Ionosphere (*D*=34)	7	NA	93.45	93.45	93.45	94.70(95.44)	95.38(95.73)	95.50(95.73)	**95.56**(95.73)	95.50(95.73)
	14	NA	90.88	92.59	93.73	94.30(94.87)	94.93(95.73)	**95.56**(95.73)	95.21(95.73)	95.21(95.73)
	20	NA	90.03	92.02	92.88	93.79(94.30)	93.90(94.30)	**94.19**(94.30)	93.73(94.02)	94.13(94.30)
	27	NA	89.17	91.17	90.88	91.17(91.45)	**91.45**(91.45)	**91.45**(91.45)	**91.45**(91.45)	**91.45**(91.45)
Satellite (*D*=36)	7	NA	86.85	88.20	**88.55**	87.89(88.10)	88.25(88.55)	88.44(88.55)	**88.55**(88.55)	88.46(88.55)
	14	NA	89.45	89.85	90.10	90.61(90.85)	**90.88**(90.95)	90.87(91.00)	90.80(90.95)	90.82(90.95)
	22	NA	90.45	91.10	**91.45**	91.36(91.45)	91.37(91.45)	91.44(91.45)	**91.45**(91.45)	91.41(91.45)
	29	NA	90.40	90.70	90.95	91.10(91.25)	91.21(91.25)	**91.24**(91.25)	**91.24**(91.25)	91.18(91.25)
Sonar (*D*=60)	12	NA	87.02	89.42	92.31	92.40(93.75)	93.65(94.71)	94.71(95.67)	94.61(95.19)	**94.81**(95.67)
	24	NA	89.90	90.87	93.75	95.49(95.67)	95.86(96.63)	95.96(96.63)	**96.34**(97.12)	96.15(97.12)
	36	NA	88.46	91.83	93.27	95.09(95.67)	95.67(96.15)	**95.82**(96.15)	95.67(96.15)	95.67(96.15)
	48	NA	91.82	92.31	91.35	92.02(92.79)	92.60(92.79)	**93.17**(93.27)	**93.17**(93.27)	93.08(93.27)
Numeral (*D*=100)	20	NA	86.05	89.05	89.45	88.86(89.70)	89.75(90.20)	90.05(90.20)	90.16(90.60)	**90.18**(90.50)
	40	NA	94.40	94.90	95.15	94.82(95.00)	95.03(95.10)	95.38(95.75)	**95.74**(96.00)	95.53(95.80)
	60	NA	95.50	96.05	96.05	96.05(96.20)	96.39(96.55)	96.53(96.70)	**96.62**(96.70)	96.57(96.65)
	80	NA	95.55	95.80	95.80	96.09(96.30)	96.18(96.35)	**96.41**(96.50)	96.34(96.40)	96.38(96.55)

* The four values are 1/5, 2/5, 3/5, and 4/5 of *D*.
** For GAs, *x* and *y* in *x*(*y*) represent the average and maximum rates, respectively.
*** Bold typefaces emphasize the best solutions in each row.

increase linearly with d because the local search operations vary from generation to generation.

6.4 Convergences and Comparisons of GAs

The aim of this section is twofold: to provide an analysis of the convergence characteristics of the GAs and a more rigorous comparison of the GAs and SFFS. To make the computation time for the algorithms comparable, the stopping condition was modified to use the same number of pure atomic operations rather than the same number of generations. For each value of d, we set the unit amount of pure atomic operations—denoted by N_{atom}—to be similar to the amount used in the experiments shown in Table 3. The recognition rates were observed at six positions, as shown in Tables 5, 6, 7, and 8. The early generations (i.e., $1/8 \sim 1/2\ N_{atom}$) allows us to compare the GAs and SFFS with similar computation times. As before, the averages and maxima of five runs are presented. The best solutions are shown in bold typefaces.

We compared five GAs at the six time positions. Most of the best solutions were found by HGAs over the four data sets. $\mathrm{HGAs}(r \geq 2)$ were better than HGA(1) in finding the best solutions. Regarding the convergence, $\mathrm{HGAs}(r \geq 2)$ showed more consistent improvements than SGA and HGA(1), particularly after the time $1N_{atom}$.

TABLE 4
Computation Times (Unit: Number of Atomic Operations and %)

Dataset	d	ES	SFS	PTA	SFFS	SGA	HGA(1)	HGA(2)	HGA(3)	HGA(4)
Vehicle (D=18)	4	12.2K	160(0)	0.41K(26)	6.00K(23)	40K(91)	36K(89)	36K(88)	37K(88)	36K(83)
	7	223K	392(0)	1.02K(30)	6.00K(23)	70K(81)	65K(75)	65K(75)	66K(81)	63K(76)
	11	350K	748(0)	2.13K(31)	6.00K(23)	110K(81)	102K(75)	101K(77)	101K(79)	95K(77)
	14	42.8K	980(0)	3.12K(29)	6.00K(23)	140K(93)	128K(88)	127K(89)	122K(88)	115K(89)
WDBC (D=30)	6	3.56M	0.56K(0)	1.37K(36)	25.2K(12)	120K(65)	90K(61)	94K(63)	98K(61)	94K(58)
	12	1.04G	1.77K(0)	4.47K(41)	25.2K(12)	240K(6)	227K(9)	231K(25)	215K(34)	212K(35)
	18	1.56G	3.19K(0)	8.86K(32)	25.2K(12)	360K(3)	359K(3)	342K(24)	351K(34)	332K(37)
	24	14.3M	4.40K(0)	14.1K(30)	25.2K(12)	480K(47)	386K(45)	376K(52)	399K(58)	353K(57)
Sonar (D=60)	12	$1.68\text{x}10^{13}$	4.11K(0)	9.51K(24)	0.32M(16)	0.48M(40)	0.40M(22)	0.42M(28)	0.44M(25)	0.42M(27)
	24	$8.65\text{x}10^{17}$	13.4K(0)	32.8K(27)	0.32M(16)	0.96M(0.4)	1.05M(0.4)	1.07M(12)	1.14M(22)	1.11M(30)
	36	$1.30\text{x}10^{18}$	24.4K(0)	66.5K(25)	0.32M(16)	1.44M(0.8)	1.57M(0.3)	1.57M(12)	1.66M(21)	1.60M(28)
	48	$6.72\text{x}10^{13}$	33.7K(0)	107K(21)	0.32M(16)	1.92M(35)	1.65M(16)	1.64M(33)	1.78M(33)	1.63M(33)
Numeral (D=100)	20	$1.07\text{x}10^{22}$	18.3K(0)	41.3K(28)	2.68M(14)	2.00M(15)	1.22M(3)	1.31M(10)	1.30M(16)	1.29M(17)
	40	$5.50\text{x}10^{29}$	60.7K(0)	147K(32)	2.68M(14)	4.00M(0)	3.40M(0)	3.42M(4)	3.57M(8)	3.45M(10)
	60	$8.25\text{x}10^{29}$	111K(0)	300K(25)	2.68M(14)	6.00M(0)	5.34M(0)	5.24M(5)	5.22M(9)	4.99M(10)
	80	$4.29\text{x}10^{22}$	153K(0)	485K(20)	2.68M(14)	8.00M(14)	5.30M(1)	5.51M(10)	5.75M(13)	5.68M(13)

Next, we compare the GAs with SFFS under the condition that the computation time be the same. The last column shows the accuracy rates produced by SFFS with the same time budgets as the GAs.

We have 16 cases, four for each of the four data sets. The SFFS produced $1 \sim 4$ solutions for them. The marks "NA" and "-" represent the situations of SFFS execution, corresponding to "solution not found yet" and "execution already ended," respectively. We observed that no improvement was afforded by SFFS after the first solution, except in three cases: $d = 6$ for WDBC, $d = 27$ for Ionosphere, and $d = 36$ for Sonar. Although SFFS adds and removes features dynamically, it is prone to become stuck at a local optimum point, due to its highly localized search nature. In 8 out of 16 cases, SFFS led all the five GAs in the early stages, in terms of the average rates. As time went by, however, the GAs turned out to be comparable or superior to SFFS.

We also counted the numbers of wins, ties, and losses for each of the GAs versus SFFS, for additional information. There are a total of 41 solutions produced by SFFS in Tables 5, 6, 7, and 8. As an example, HGA(2) recorded 24 wins, 3 ties, and 14 losses. The ratio of wins to losses was 63 percent. HGA(3) and HGA(4) showed similar ratios of 66 and 61 percent, respectively. HGA(1) was comparable to SFFS with a ratio of 50 percent. Based on the above results, it would appear that the HGAs provide better overall performance than SFFS, in the case where the computation time is the same.

6.5 Comparisons of GAs and Multistart Algorithms

The multistart algorithm (MS) is a heuristic that repeats the "random initial solution + local optimization" mechanism a number of times and returns the best solution found. Here, we compare MS with HGA, in order to determine how much the *genetic process* contributes to the search, in addition to the search power provided by the local optimization. To accomplish this, MS has to use the same local optimization and the same time budget as that used in HGA. The structure of the multistart algorithm is as follows:

```
Algorithm multistart:
1.  $T = 0$;
2.  **repeat** {
3.    $X = \Phi$; $Y = \Phi$;
4.    **for**($i = 1$; $i <= D$; $i++$) **if**(random_uniform() $< d/D$)
          $\mathrm{X} = \mathrm{X} \cup \{i\}$; **else** $\mathrm{Y} = \mathrm{Y} \cup \{i\}$;
5.    **switch**{
6.      **case** $|\mathrm{X}| = d$: $ripple\_rem(r); ripple\_add(r)$;
7.      **case** $|\mathrm{X}| < d$: **repeat** $ripple\_add(r)$ $d - |\mathrm{X}|$ times;
8.      **case** $|\mathrm{X}| > d$: **repeat** $ripple\_rem(r)$ $|\mathrm{X}| - d$ times;
9.    }
10.   $Jx = \mathrm{J}(\mathrm{X})$;
11.   **if** ($Jx > T$) $T = Jx$;
12. **until** (time limitation is reached);
13. report $T$ as accuracy of the final solution;
```

Step 4 is the same as the statement used by the GAs for generating the initial population. Steps 5 through 9 are the same as the local search used by the function local_improvement() in the HGA of Section 4.

We ran MS with two ripple factors, $r = 2$ and 3. $\mathrm{MS}(r)$ refers to the MS with ripple factor r. Table 9 presents the average and maximum rates from five runs. The data for HGA(2) and HGA(3) were copied from Table 3 for comparison.

HGA and MS were comparable for the smallest problem, WDBC. HGA outperformed MS for all the other problems. It is notable that the performance gap between them increased as the problem size grew. This gap reflects the contribution of the genetic process.

6.6 Discussion

Based on the experimental results and analyses, we draw a number of conclusions. It should be noted that the

TABLE 5
Convergence of GAs for WDBC (Recognition Rate in %)

d	Algorithms / Time	SGA	HGA(1)	HGA(2)	HGA(3)	HGA(4)	SFFS*
6 (N_{atom}=50K)	(1/8)N_{atom}	93.67(94.20)	94.09(94.20)	**94.20**(94.20)	93.67(93.67)	93.99(94.20)	94.02
	(1/4)N_{atom}	93.74(94.20)	94.09(94.20)	94.20(94.20)	93.78(94.20)	**94.34**(94.90)	94.20
	(1/2)N_{atom}	93.78(94.20)	94.09(94.20)	94.20(94.20)	93.78(94.20)	**94.34**(94.90)	94.20
	1N_{atom}	93.78(94.20)	94.09(94.20)	94.20(94.20)	93.88(94.20)	**94.34**(94.90)	-
	2N_{atom}	93.78(94.20)	94.09(94.20)	94.20(94.20)	93.99(94.20)	**94.34**(94.90)	-
	4N_{atom}	93.78(94.20)	94.09(94.20)	94.20(94.20)	94.20(94.20)	**94.48**(94.90)	-
12 (N_{atom}=100K)	(1/8)N_{atom}	93.99(94.20)	93.99(94.20)	93.95(94.38)	93.85(93.85)	93.99(94.38)	**94.20**
	(1/4)N_{atom}	94.06(94.38)	94.06(94.20)	93.95(94.38)	93.92(94.20)	94.06(94.38)	**94.20**
	(1/2)N_{atom}	94.06(94.38)	94.09(94.38)	94.06(94.38)	93.95(94.38)	94.06(94.38)	-
	1N_{atom}	94.06(94.38)	94.17(94.38)	94.06(94.38)	94.06(94.38)	**94.27**(94.38)	-
	2N_{atom}	94.06(94.38)	94.17(94.38)	94.17(94.38)	94.06(94.38)	**94.27**(94.38)	-
	4N_{atom}	94.06(94.38)	94.17(94.38)	94.17(94.38)	94.06(94.38)	**94.27**(94.38)	-
18 (N_{atom}=150K)	(1/8)N_{atom}	93.85(93.85)	93.88(94.20)	93.92(94.20)	93.99(94.20)	93.92(94.20)	**94.20**
	(1/4)N_{atom}	93.85(93.85)	93.99(94.20)	93.92(94.20)	93.99(94.20)	93.92(94.20)	**94.20**
	(1/2)N_{atom}	93.85(93.85)	**93.99**(94.20)	93.92(94.20)	**93.99**(94.20)	**93.99**(94.20)	-
	1N_{atom}	93.85(93.85)	**93.99**(94.20)	93.92(94.20)	**93.99**(94.20)	**93.99**(94.20)	-
	2N_{atom}	93.85(93.85)	93.99(94.20)	93.92(94.20)	**94.06**(94.20)	**94.06**(94.20)	-
	4N_{atom}	93.85(93.85)	93.99(94.20)	93.99(94.20)	94.06(94.20)	**94.20**(94.20)	-
24 (N_{atom}=200K)	(1/8)N_{atom}	**93.85**(93.85)	**93.85**(93.85)	**93.85**(93.85)	93.43(93.85)	93.39(93.85)	**93.85**
	(1/4)N_{atom}	**93.85**(93.85)	**93.85**(93.85)	**93.85**(93.85)	**93.85**(93.85)	**93.85**(93.85)	-
	(1/2)N_{atom}	**93.85**(93.85)	**93.85**(93.85)	**93.85**(93.85)	**93.85**(93.85)	**93.85**(93.85)	-
	1N_{atom}	**93.85**(93.85)	**93.85**(93.85)	**93.85**(93.85)	**93.85**(93.85)	**93.85**(93.85)	-
	2N_{atom}	**93.85**(93.85)	**93.85**(93.85)	**93.85**(93.85)	**93.85**(93.85)	**93.85**(93.85)	-
	4N_{atom}	**93.85**(93.85)	**93.85**(93.85)	**93.85**(93.85)	**93.85**(93.85)	**93.85**(93.85)	-

* NA: solution not found yet, -: execution already ended (valid through Tables 5-8).

TABLE 6
Convergence of GAs for Ionosphere (Recognition Rate in %)

d	Algorithms / Time	SGA	HGA(1)	HGA(2)	HGA(3)	HGA(4)	SFFS
7 (N_{atom}=50K)	(1/8)N_{atom}	94.02(94.87)	94.53(94.87)	**94.87**(95.44)	94.19(94.87)	94.25(94.87)	93.45
	(1/4)N_{atom}	94.53(95.73)	94.99(95.73)	95.04(95.44)	**95.21**(95.73)	94.99(95.73)	93.45
	(1/2)N_{atom}	94.87(95.73)	95.16(95.73)	95.21(95.44)	95.2195.73)	**95.44**(95.73)	93.45
	1N_{atom}	94.87(95.73)	95.38(95.73)	95.38(95.73)	**95.61**(95.73)	95.56(95.73)	93.45
	2N_{atom}	94.87(95.73)	95.56(95.73)	95.61(95.73)	**95.67**(95.73)	95.56(95.73)	-
	4N_{atom}	94.87(95.73)	**95.67**(95.73)	95.61(95.73)	**95.67**(95.73)	**95.67**(95.73)	-
14 (N_{atom}=100K)	(1/8)N_{atom}	92.59(93.45)	93.16(93.73)	93.28(93.73)	93.50(94.02)	93.33(94.30)	**93.73**
	(1/4)N_{atom}	93.79(94.30)	93.68(94.02)	**94.19**(94.87)	93.85(94.59)	94.13(94.87)	93.73
	(1/2)N_{atom}	94.13(94.87)	94.19(94.59)	**94.70**(95.16)	94.42(94.87)	94.47(94.87)	93.73
	1N_{atom}	94.35(94.87)	94.42(94.87)	**95.16**(95.73)	95.04(95.73)	94.70(94.87)	-
	2N_{atom}	94.42(94.87)	94.93(95.73)	**95.38**(95.73)	**95.38**(95.73)	94.93(95.73)	-
	4N_{atom}	94.42(94.87)	94.93(95.73)	95.38(95.73)	**95.73**(95.73)	95.38(95.73)	-
20 (N_{atom}=150K)	(1/8)N_{atom}	91.85(92.31)	92.42(93.16)	92.36(92.59)	92.25(92.88)	92.76(93.16)	**92.88**
	(1/4)N_{atom}	92.08(92.31)	92.82(93.16)	92.99(93.73)	93.11(94.02)	**93.28**(94.30)	92.88
	(1/2)N_{atom}	92.59(93.16)	92.99(93.73)	**93.39**(93.73)	93.33(94.02)	93.33(94.30)	92.88
	1N_{atom}	93.11(93.73)	93.39(94.02)	93.62(94.02)	93.56(94.02)	**93.73**(94.30)	-
	2N_{atom}	93.28(93.73)	93.73(94.02)	**94.13**(94.30)	93.73(94.30)	94.07(94.30)	-
	4N_{atom}	93.39(93.73)	93.73(94.02)	**94.13**(94.30)	93.96(94.30)	**94.13**(94.30)	-
27 (N_{atom}=200K)	(1/8)N_{atom}	90.88(91.45)	90.66(91.17)	**91.17**(91.45)	91.05(91.45)	**91.17**(91.45)	90.60
	(1/4)N_{atom}	91.00(91.45)	**91.34**(91.45)	91.28(91.45)	91.23(91.45)	91.28(91.45)	90.88
	(1/2)N_{atom}	91.23(91.45)	**91.45**(91.45)	91.40(91.45)	91.40(91.45)	**91.45**(91.45)	-
	1N_{atom}	91.28(91.45)	**91.45**(91.45)	**91.45**(91.45)	**91.45**(91.45)	**91.45**(91.45)	-
	2N_{atom}	91.34(91.45)	**91.45**(91.45)	**91.45**(91.45)	**91.45**(91.45)	**91.45**(91.45)	-
	4N_{atom}	91.34(91.45)	**91.45**(91.45)	**91.45**(91.45)	**91.45**(91.45)	**91.45**(91.45)	-

experimental results and analyses from which we draw our conclusions were based on various standard data sets covering a large spectrum of problem sizes.

1. The SFFS is the best among the sequential search algorithms.
2. The GAs, including the SGA and HGAs, outperform SFFS regardless of the problem size.
3. The HGAs outperform both SFFS and SGA. The ripple factor $r \geq 2$ is recommended.
4. The HGAs are somewhat more attractive for large-sized problems.

It would be worthwhile discussing these conclusions in more detail. It is interesting to compare our conclusions with the ones made by other comparative studies [5], [9], [12]. The first conclusion is the same as that made in conventional studies. However, the second conclusion is inconsistent with these previous results. Ferri et al. [5] insisted that SFFS and GA are comparable, but that the GA becomes inferior to SFFS as the problem size increases.

TABLE 7
Convergence of GAs for Satellite (Recognition Rate in %)

d	Algorithms / Time	SGA	HGA(1)	HGA(2)	HGA(3)	HGA(4)	SFFS
7 (N_{atom}=50K)	(1/8)N_{atom}	87.48(87.95)	87.46(88.05)	87.71(87.95)	87.64(88.20)	87.53(87.75)	**88.55**
	(1/4)N_{atom}	87.63(87.95)	87.80(88.10)	88.12(88.55)	87.89(88.20)	87.77(88.55)	**88.55**
	(1/2)N_{atom}	87.63(87.95)	88.15(88.55)	88.36(88.55)	88.06(88.55)	88.11(88.55)	**88.55**
	1N_{atom}	87.81(88.00)	88.31(88.55)	88.41(88.55)	88.31(88.55)	88.29(88.55)	**88.55**
	2N_{atom}	87.81(88.00)	88.31(88.55)	**88.55**(88.55)	**88.55**(88.55)	**88.55**(88.55)	-
	4N_{atom}	87.91(88.20)	88.40(88.55)	**88.55**(88.55)	**88.55**(88.55)	**88.55**(88.55)	-
14 (N_{atom}=100K)	(1/8)N_{atom}	89.83(90.3)	89.81(90.05)	**90.15**(90.45)	90.13(90.45)	90.09(90.65)	90.10
	(1/4)N_{atom}	89.97(90.30)	90.25(90.60)	90.39(90.60)	90.22(90.45)	**90.41**(90.65)	90.10
	(1/2)N_{atom}	90.26(90.45)	90.54(90.60)	90.63(90.85)	90.66(90.85)	**90.75**(90.80)	90.10
	1N_{atom}	90.45(90.60)	90.63(90.85)	90.72(90.90)	**90.81**(90.95)	90.78(90.90)	-
	2N_{atom}	90.46(90.60)	90.70(90.85)	90.89(90.95)	**90.90**(90.95)	90.80(90.90)	-
	4N_{atom}	90.46(90.60)	90.75(90.85)	90.90(90.95)	**90.92**(90.95)	90.90(90.95)	-
22 (N_{atom}=150K)	(1/8)N_{atom}	90.73(91.25)	**90.90**(91.10)	**90.90**(91.20)	90.88(91.20)	90.73(90.90)	NA
	(1/4)N_{atom}	90.91(91.35)	91.02(91.45)	91.13(91.45)	91.06(91.30)	91.16(91.35)	**91.45**
	(1/2)N_{atom}	91.12(91.35)	91.15(91.45)	91.20(91.45)	91.17(91.35)	91.37(91.45)	**91.45**
	1N_{atom}	91.13(91.35)	91.31(91.45)	91.35(91.45)	91.34(91.40)	**91.38**(91.45)	-
	2N_{atom}	91.22(91.45)	91.33(91.45)	**91.41**(91.45)	91.40(91.45)	91.40(91.45)	-
	4N_{atom}	91.31(91.45)	91.37(91.45)	**91.45**(91.45)	**91.45**(91.45)	91.44(91.45)	-
29 (N_{atom}=200K)	(1/8)N_{atom}	90.77(90.95)	90.83(91.00)	90.90(91.00)	**91.10**(91.25)	90.78(90.95)	NA
	(1/4)N_{atom}	90.86(91.10)	90.98(91.10)	90.98(91.10)	**91.17**(91.25)	91.05(91.15)	90.95
	(1/2)N_{atom}	90.93(91.10)	91.05(91.20)	91.19(91.25)	**91.20**(91.25)	91.10(91.20)	-
	1N_{atom}	90.97(91.10)	91.19(91.25)	**91.25**(91.25)	91.24(91.25)	91.18(91.25)	-
	2N_{atom}	91.10(91.25)	91.19(91.25)	**91.25**(91.25)	91.24(91.25)	91.19(91.25)	-
	4N_{atom}	91.13(91.25)	91.20(91.25)	**91.25**(91.25)	91.24(91.25)	**91.25**(91.25)	-

TABLE 8
Convergence of GAs for Sonar (Recognition Rate in %)

d	Algorithms / Time*	SGA	HGA(1)	HGA(2)	HGA(3)	HGA(4)	SFFS
12 (N_{atom}=0.3M)	(1/8)N_{atom}	92.79(94.23)	91.73(92.79)	**93.17**(94.23)	91.83(92.31)	92.40(93.75)	92.31
	(1/4)N_{atom}	**93.85**(95.19)	92.40(92.79)	93.75(94.23)	92.69(93.27)	92.69(93.75)	92.31
	(1/2)N_{atom}	94.04(95.19)	93.17(93.75)	**94.23**(94.71)	**94.23**(95.19)	93.56(94.71)	92.31
	1N_{atom}	94.23(95.67)	93.65(95.67)	94.71(95.19)	**95.00**(95.67)	94.13(94.71)	92.31
	2N_{atom}	94.23(95.67)	93.85(95.67)	94.90(95.67)	**95.19**(95.67)	94.71(95.67)	-
	4N_{atom}	94.23(95.67)	93.94(95.67)	94.90(95.67)	**95.29**(95.67)	94.90(95.67)	-
24 (N_{atom}=0.6M)	(1/8)N_{atom}	92.79(93.75)	93.08(94.23)	92.98(95.19)	93.46(94.23)	92.98(94.23)	**93.75**
	(1/4)N_{atom}	93.94(94.71)	93.65(94.71)	93.94(95.19)	**94.13**(95.19)	94.04(94.23)	93.75
	(1/2)N_{atom}	95.00(95.67)	94.13(94.71)	**95.58**(96.63)	95.00(95.67)	94.62(95.19)	93.75
	1N_{atom}	95.58(96.63)	95.29(95.67)	96.25(97.12)	95.67(96.63)	**96.35**(97.12)	-
	2N_{atom}	95.67(97.12)	95.96(97.12)	**96.63**(97.12)	96.25(97.12)	**96.63**(97.12)	-
	4N_{atom}	95.67(97.12)	96.35(97.12)	**96.73**(97.12)	96.54(97.12)	**96.73**(97.12)	-
36 (N_{atom}=0.9M)	(1/8)N_{atom}	92.21(93.75)	**92.98**(94.23)	92.69(94.23)	92.31(93.75)	92.31(93.27)	91.83
	(1/4)N_{atom}	93.65(94.23)	**94.33**(95.67)	93.75(94.71)	93.08(94.23)	93.37(94.71)	93.27
	(1/2)N_{atom}	94.52(95.67)	**95.10**(95.67)	94.42(95.19)	94.42(95.19)	94.71(95.19)	93.27
	1N_{atom}	94.81(96.15)	95.48(96.15)	95.19(96.15)	**95.58**(96.12)	94.90(95.67)	-
	2N_{atom}	94.81(96.15)	95.77(96.63)	95.77(97.12)	**96.35**(97.12)	96.25(97.12)	-
	4N_{atom}	95.10(96.15)	95.96(97.12)	96.06(97.12)	**96.54**(97.12)	96.44(97.12)	-
48 (N_{atom}=1.2M)	(1/8)N_{atom}	90.87(92.31)	91.63(92.31)	**91.83**(92.31)	91.44(92.31)	91.06(91.83)	NA
	(1/4)N_{atom}	91.15(92.79)	**92.40**(93.27)	92.12(92.79)	**92.40**(92.79)	92.02(92.79)	91.35
	(1/2)N_{atom}	91.44(92.79)	**92.79**(93.75)	92.69(93.27)	**92.79**(93.27)	92.60(92.79)	-
	1N_{atom}	92.12(93.75)	92.98(93.75)	**93.27**(93.27)	92.88(93.27)	92.88(93.27)	-
	2N_{atom}	92.12(93.75)	92.98(93.75)	**93.27**(93.27)	93.08(93.27)	93.08(93.75)	-
	4N_{atom}	92.12(93.75)	92.98(93.75)	**93.37**(93.75)	93.27(93.75)	93.27(93.75)	-

Jain et al. [9] concluded that the GA always gives rise to premature convergence and is inferior to SFFS. The work by Kudo and Sklansky in [12] presented a recommendation that SFFS is suitable for small and medium-sized problems, while the GA is more suitable for large-sized problems.

The inconsistency among these different studies seems to be due to the difference in the specifications of the GA used. As mentioned in Section 1, GA has many inherent variations and parameters that need to be handled properly in the implementation stage, in order for reasonable or good performance to be obtained. In order to maximize the reproducibility and minimize the ambiguity in the GA implementation, we presented rather detailed specifications for the GAs in Sections 3 and 4.

The SFFS consumed the longest computation time among the sequential search algorithms. The GAs required more time than the sequential search algorithms for their convergence. A good way of accelerating GAs is to use parallel computation since the local search operations, as well as the genetic process have abundant parallelism. For example, to execute an instance of the local search operations, rem^g, we may generate all the candidate subsets and then evaluate them in parallel, and finally choose the

TABLE 9
Comparisons of GAs and Multistart Algorithms (Unit: %)

Datasets	*d*	HGA(2)	HGA(3)	MS(2)*	MS(3)*
WDBC (*D*=30)	6	94.38(94.90)	93.99(94.20)	94.20(94.20)	**94.48**(94.90)
	12	**94.06**(94.38)	**94.06**(94.38)	93.99(94.38)	94.02(94.38)
	18	93.99(94.20)	**94.13**(94.20)	**94.13**(94.20)	94.06(94.20)
	24	**93.85**(93.85)	**93.85**(93.85)	**93.85**(93.85)	**93.85**(93.85)
Ionosphere (*D*=34)	7	95.50(95.73)	**95.56**(95.73)	95.10(95.44)	95.16(95.44)
	14	**95.56**(95.73)	95.21(95.73)	94.42(94.59)	94.19(94.59)
	20	**94.19**(94.30)	93.73(94.02)	93.28(93.45)	92.99(93.45)
	27	**91.45**(91.45)	**91.45**(91.45)	**91.45**(91.45)	91.40(91.45)
Satellite (*D*=36)	7	88.44(88.55)	**88.55**(88.55)	88.19(88.55)	88.43(88.55)
	14	**90.87**(91.00)	90.80(90.95)	90.45(90.55)	90.63(90.90)
	22	91.44(91.45)	**91.45**(91.45)	91.16(91.45)	91.22(91.35)
	29	**91.24**(91.25)	**91.24**(91.25)	91.16(91.25)	91.17(91.25)
Sonar (*D*=60)	12	**94.71**(95.67)	94.61(95.19)	93.37(94.23)	92.98(94.23)
	24	95.96(96.63)	**96.34**(97.12)	93.94(94.71)	94.04(94.71)
	36	**95.82**(96.15)	95.67(96.15)	93.37(93.75)	93.65(94.23)
	48	**93.17**(93.27)	**93.17**(93.27)	91.63(92.31)	91.73(92.31)
Numeral (*D*=100)	20	90.05(90.20)	**90.16**(90.60)	89.02(89.55)	89.47(89.85)
	40	95.38(95.75)	**95.74**(96.00)	94.68(95.30)	94.54(94.75)
	60	96.53(96.70)	**96.62**(96.70)	95.88(96.10)	96.17(96.25)
	80	**96.41**(96.50)	96.34(96.40)	95.88(96.05)	96.15(96.35)

* MS(r) represents the multistart algorithm with ripple factor r.

least significant one. Given a sufficient number of processors, all of the local search operations can be accomplished in constant time, independent of the sizes of X and Y.

The GAs are very advantageous compared to the deterministic algorithms, in the sense that further improvements can be made in various ways. Although, in Section 3.6, the same set of parameter values was used for all of the data sets, tuning the genetic parameters for a particular data set may lead to an improvement. As shown in Section 6.4, the convergence characteristics of $\mathrm{HGA}(r \geq 2)$ encourage us to go further in order to come closer to the optimum point. Performing multiple executions and then choosing the best solution may also be another way of improving this algorithm. Note that the use of such methods is not possible in the deterministic algorithms.

7 Concluding Remarks

Novel hybrid GAs were proposed to solve the feature selection problem, with the goal of achieving leading-edge performance over the conventional algorithms. Local search operations parameterized with ripple factors were devised and embedded in the HGAs. The experimental results obtained for a variety of standard data sets revealed that the stated objective was successfully accomplished.

Our contributions can be summarized as follows: Significant improvements were observed through the proposed hybrid GAs. This will contribute to the performance improvements of various applications requiring feature selection. Another advantage offered by the hybridization was the acquisition of subset size control. The concept of atomic operations has proven to be useful in rigorously analyzing and comparing the timing efficiencies of the algorithms.

In the future, it would be worthwhile developing other schemes, such as gene rearrangement for chromosomal encoding or more suitable genetic operators. Analyzing the effect of varying population sizes and the further tuning of other genetic parameters leave some room for further improvement. Speeding up the processing of local search operations is another important issue.

Acknowledgments

The authors would like to thank the anonymous reviewers for their constructive comments. This work was partly supported by grant No. (R01-2003-000-10879-0) from the Basic Research Program of the Korea Science and Engineering Foundation, and by Brain Korea 21 Project.

References

[1] F.Z. Brill, D.E. Brown, and W.N. Martin, "Fast Genetic Selection of Features for Neural Network Classifiers," *IEEE Trans. Neural Networks,* vol. 3, no. 2, pp. 324-328, Mar. 1992.

[2] T.N. Bui and B.R. Moon, "Genetic Algorithm and Graph Partitioning," *IEEE Trans. Computers,* vol. 45, no. 7, pp. 841-855, July 1996.

[3] M. Dash and H. Liu, "Feature Selection for Classification," *Intelligent Data Analysis,* vol. 1, no. 3, pp. 131-156, 1997.

[4] L. Davis, *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, 1991.

[5] F.J. Ferri, P. Pudil, M. Hatef, and J. Kittler, "Comparative Study of Techniques for Large-Scale Feature Selection," *Pattern Recognition in Practice IV,* E.S. Gelsema and L.N. Kanal, eds., pp. 403-413, 1994.

[6] J. Holland, *Adaptation in Nature and Artificial Systems.* MIT Press, 1992.

[7] C.-C. Hung, A. Fahsi, W. Tadesse, and T. Coleman, "A Comparative Study of Remotely Sensed Data Classification Using Principal Components Analysis and Divergence," *Proc. IEEE Int'l Conf. Systems, Man, and Cybernetics,* pp. 2444-2449, 1997.

[8] H. Ishikawa, “Multiscale Feature Selection in Stereo,” *Proc. IEEE Int'l Conf. Computer Vision and Pattern Recognition,* pp. 132-137, 1999.
[9] A. Jain and D. Zongker, “Feature Selection: Evaluation, Application, and Small Sample Performance,” *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 19, no. 2, pp. 153-158, Feb. 1997.
[10] P. Jog, J. Suh, and D. Gucht, “The Effect of Population Size, Heuristic Crossover and Local Improvement on a Genetic Algorithm for the Traveling Salesman Problem,” *Proc. Int'l Conf. Genetic Algorithms,* pp. 110-115, 1989.
[11] J. Kittler, “Feature Selection and Extraction,” *Handbook of Pattern Recognition and Image Processing,* T.Y. Young and K.S. Fu, eds., pp. 59-83, 1986.
[12] M. Kudo and J. Sklansky, “Comparison of Algorithms that Select Features for Pattern Recognition,” *Pattern Recognition,* vol. 33, no. 1, pp. 25-41, 2000.
[13] L.I. Kuncheva and L.C. Jain, “Nearest Neighbor Classifier: Simultaneous Editing and Feature Selection,” *Pattern Recognition Letters,* vol. 20, pp. 1149-1156, 1999.
[14] P. Langley, “Selection of Relevant Features in Machine Learning,” *Proc. AAAI Fall Symp. Relevance,* pp. 1-5, 1994.
[15] *Principles of Visual Information Retrieval.* M.S. Lew, ed., Springer, 2001.
[16] Y. Liu and F. Dellaert, “A Classification Based Similarity Metric for 3D Image Retrieval,” *Proc. IEEE Int'l Conf. Computer Vision and Pattern Recognition,* pp. 800-805, 1998.
[17] M.J. Martin-Bautista and M.-A. Vila, “A Survey of Genetic Feature Selection in Mining Issues,” *Proc. 1999 Congress on Evolutionary Computation (CEC '99),* pp. 1314-1321, July 1999.
[18] K. Messer and J. Kittler, “Using Feature Selection to Aid an Iconic Search through an Image Database,” *Proc. EEE Int'l Conf. Acoustics, Speech, and Signal processing (ICASSP),* vol. 4, pp. 2605-2608, 1997.
[19] P.M. Murphy and D.W. Aha, “UCI Repository for Machine Learning Databases,” technical report, Dept. of Information and Computer Science, Univ. of California, Irvine, Calif., 1994, http://www.ics.uci.edu/mlearn/MLRepository.html.
[20] P.M. Narendra and K. Fukunaga, “A Branch and Bound Algorithm for Feature Subset Selection,” *IEEE Trans. Computers,* vol. 26, no. 9, pp. 917-922, Sept. 1977.
[21] I.-S. Oh and C.Y. Suen, “Distance Features for Neural Network-Based Recognition of Handwritten Characters,” *Int'l J. Document Analysis and Recognition,* vol. 1, no. 2, pp. 73-88, 1998.
[22] I.-S. Oh, J.-S. Lee, and C.Y. Suen, “Analysis of Class Separation and Combination of Class-Dependent Features for Handwriting Recognition,” *IEEE Trans. Pattern Analysis and Machine Intelligence,* vol. 21, no. 10, pp. 1089-1094, Oct. 1999.
[23] I.-S. Oh, J.-S. Lee, and B.-R. Moon, “Local Search-Embedded Genetic Algorithms for Feature Selection,” *Proc. Int'l Conf. Pattern Recognition,* 2002.
[24] S. Piramuthu, “Evaluating Feature Selection Methods for Learning in Data Mining Applications,” *Proc. 31st Ann. Hawaii Int'l Conf. System Science,* pp. 294-301, 1998.
[25] P. Pudil, J. Novovicova, and J. Kittler, “Floating Search Methods in Feature Selection,” *Pattern Recognition Letters,* vol. 15, pp. 1119-1125, 1994.
[26] S. Puuronen, A. Tsymbal, and I. Skrypnik, “Advanced Local Feature Selection in Medical Diagnostics,” *Proc. 13th IEEE Symp. Computer-Based Medical Systems,* pp. 25-30, 2000.
[27] M.L. Raymer, W.F. Punch, E.D. Goodman, L.A. Kuhn, and A.K. Jain, “Dimensionality Reduction Using Genetic Algorithms,” *IEEE Trans. Evolutionary Computation,* vol. 4, no. 2, pp. 164-171, July 2000.
[28] W. Siedlecki and J. Sklansky, “On Automatic Feature Selection,” *Int'l J. Pattern Recognition and Artificial Intelligence,* vol. 2, no. 2, pp. 197-220, 1988.
[29] W. Siedlecki and J. Sklansky, “A Note on Genetic Algorithms for Large-Scale Feature Selection,” *Pattern Recognition Letters,* vol. 10, pp. 335-347, 1989.
[30] J.H. Yang and V. Honavar, “Feature Subset Selection Using a Genetic Algorithm,” *IEEE Intelligent Systems,* vol. 13, no. 2, pp. 44-49, 1998.
[31] B. Yu and B. Yuan, “A More Efficient Branch and Bound Algorithm for Feature Selection,” *Pattern Recognition,* vol. 26, no. 6, pp. 883-889, 1993.
[32] X. Zheng, B.A. Julstrom, and W. Cheng, “Design of Vector Quantization Codebooks Using a Genetic Algorithm,” *Proc. IEEE Int'l Conf. Evolutionary Computation,* pp. 525-529, 1997.

Il-Seok Oh received the BS degree in computer engineering from Seoul National University, Korea, and the PhD degree in computer science from KAIST (Korea Advanced Institute of Science and Technology), Korea, in 1984 and 1992, respectively. He is currently a professor of Division of Electronics and Computer Engineering at Chonbuk National University, Jeonju, Korea. From 1996-1997, he was a visiting scientist of CENPARMI at Concordia University, Canada. He serves now as an associate editor-in-chief of the *Journal of Korea Information Science Society*. His research interests include character recognition in document images and in natural scene images, text-based retrieval of old document images, and handwriting recognition, using genetic algorithm to solve pattern recognition problems. He is a member of the IEEE and IEEE Computer Society. His web site is http://cv.chonbuk.ac.kr/~isoh.

Jin-Seon Lee received the BS degree in computer science and statistics, and the MS and PhD degrees in computer engineering from Chonbuk National University, Jeonju, Korea, in 1985, 1988, and 1995, respectively. From 1988 to 1991, she was a researcher at the Electronics and Telecommunications Research Institute, Daejeon, Korea. Currently, she is an associate professor in Department of Computer Engineering at Woosuk University, Samrye, Korea. Her research interests are the pattern recognition, image processing, and multimedia.

Byung-Ro Moon received the BS degree in computer science and statistics from Seoul National University, Seoul, Korea, the MS degree in computer science from Korea Advanced Institute of Science and Technology, Korea, and the PhD degree from the Pennsylvania State University, University Park, PA, in 1985, 1987, and 1994, respectively. Currently, he is associate professor and director of the Optimization Laboratory in the School of Computer Science and Engineering at Seoul National University, Seoul, Korea. From 1987 to 1991, he was an associate research staffer at the Central Laboratory of LG Electronics Co., Ltd., Seoul, Korea. From November 1994 to 1995, he was a postdoctoral scholar in the VLSI CAD Laboratory at the University of California at Los Angeles. From 1996 to 1997, he was a principal research staff member in the DT Research Center, LG Semicon Co., Ltd., Seoul, Korea. He joined Seoul National University in September 1997. He is also the chief executive officer of Optus Inc. He has published more than 90 technical papers and two books. His major interest is the theory and application of optimization methodologies (evolutionary computation, algorithm design/analysis, optimization modeling, etc.). The applications of optimization include combinatorial problems, data mining, personalized marketing, campaign optimization, e-commerce, search, context recognition, graph partitioning, circuit layout, scheduling, and stock prediction. He served as the publication chair of IEEE CEC'2001 and a committee member of GP'97, GP'98, ISIAC'98, and GECCO'99 through GECCO'2004. He is also a member of the council of authors of International Society for Genetic and Evolutionary Computation (ISGEC). He has provided more than 30 invited talks on genetic algorithms, optimization, CRM, education, etc. He was the recipient of a Korean Government Overseas Scholarship, in 1991-1994. He received the Best Teaching Award of the School of Computer Science and Engineering at Seoul National University, in 2001. He is a member of the IEEE.